Exhibit 4

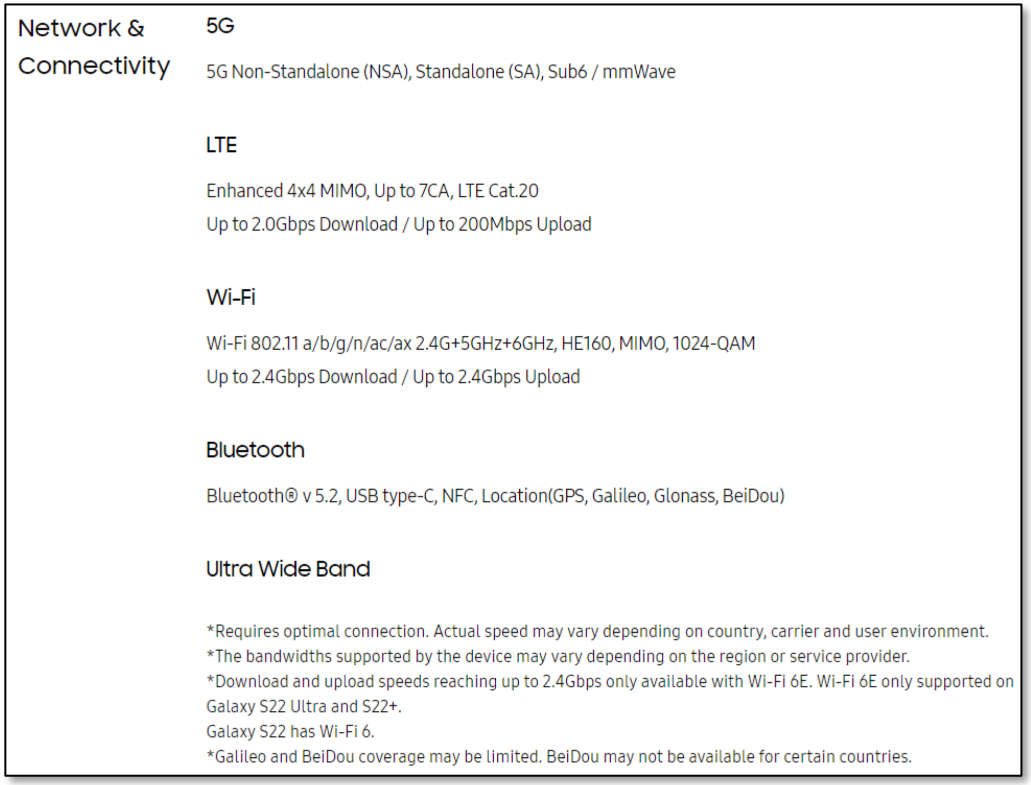
U.S. Patent No. 8,406,733 ("'733 Patent")

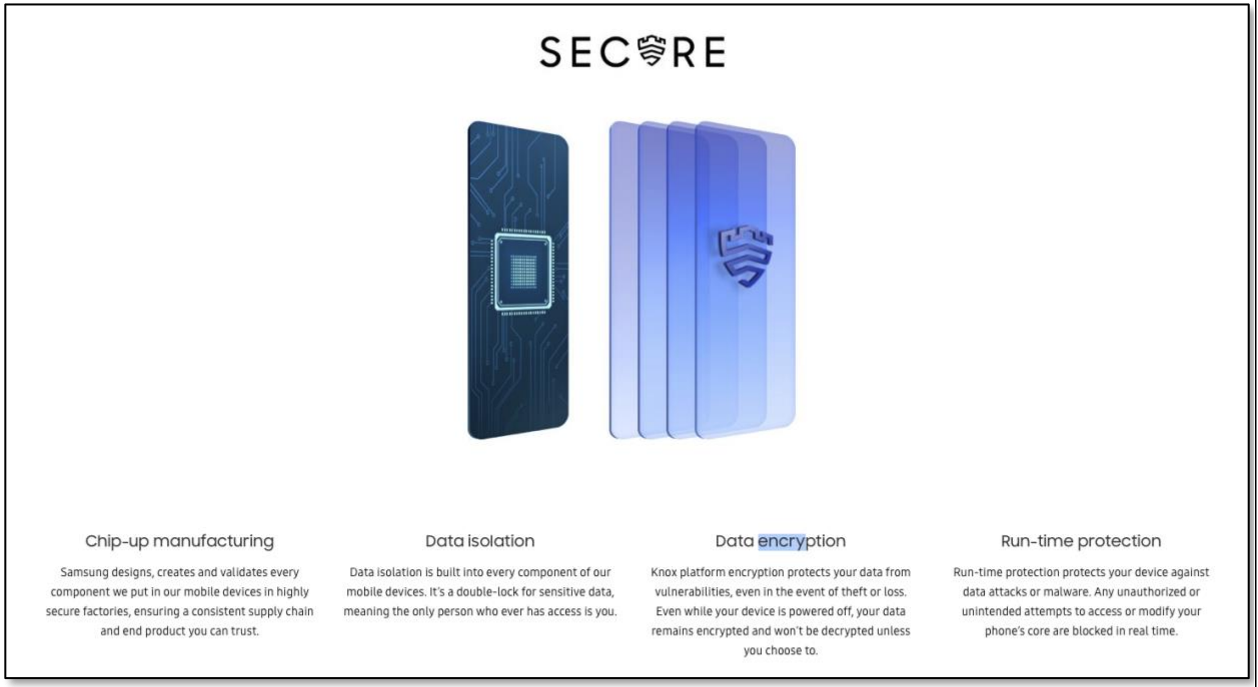
Accused Devices: Samsung's mobile electronic devices (e.g., Galaxy phones and tablets, as well as Samsung devices which include Samsung Knox functionality), and Samsung Tizen devices (e.g., TVs and wearables), and all versions and variations thereof since the issuance of the asserted patent.

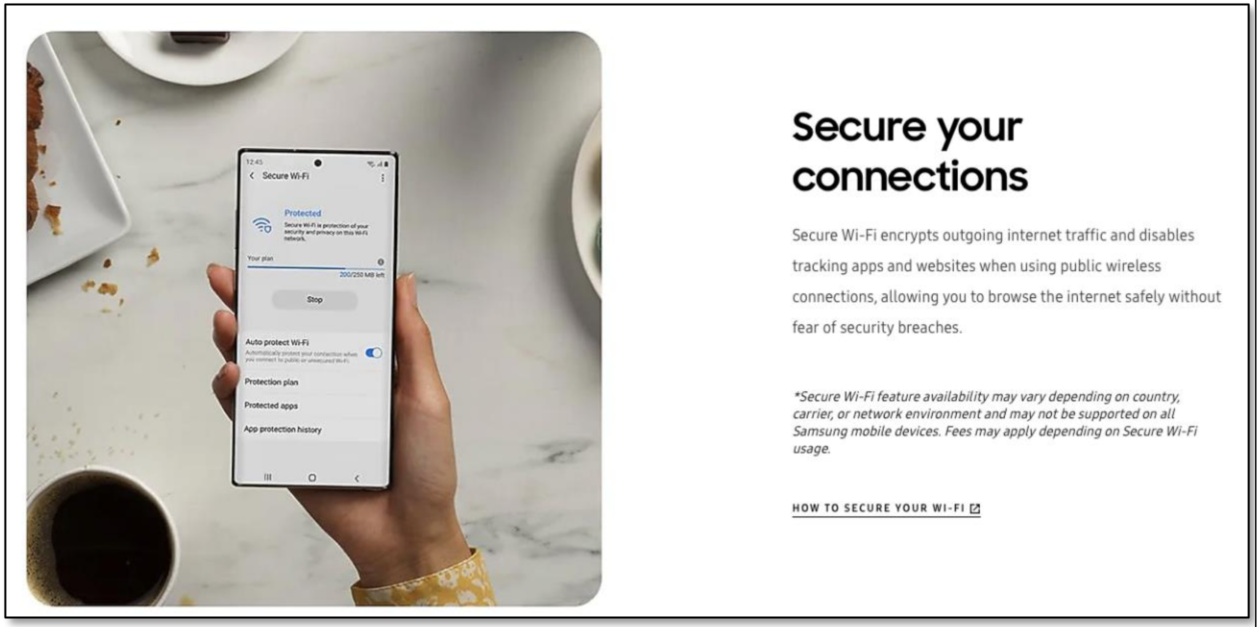
Claim 1

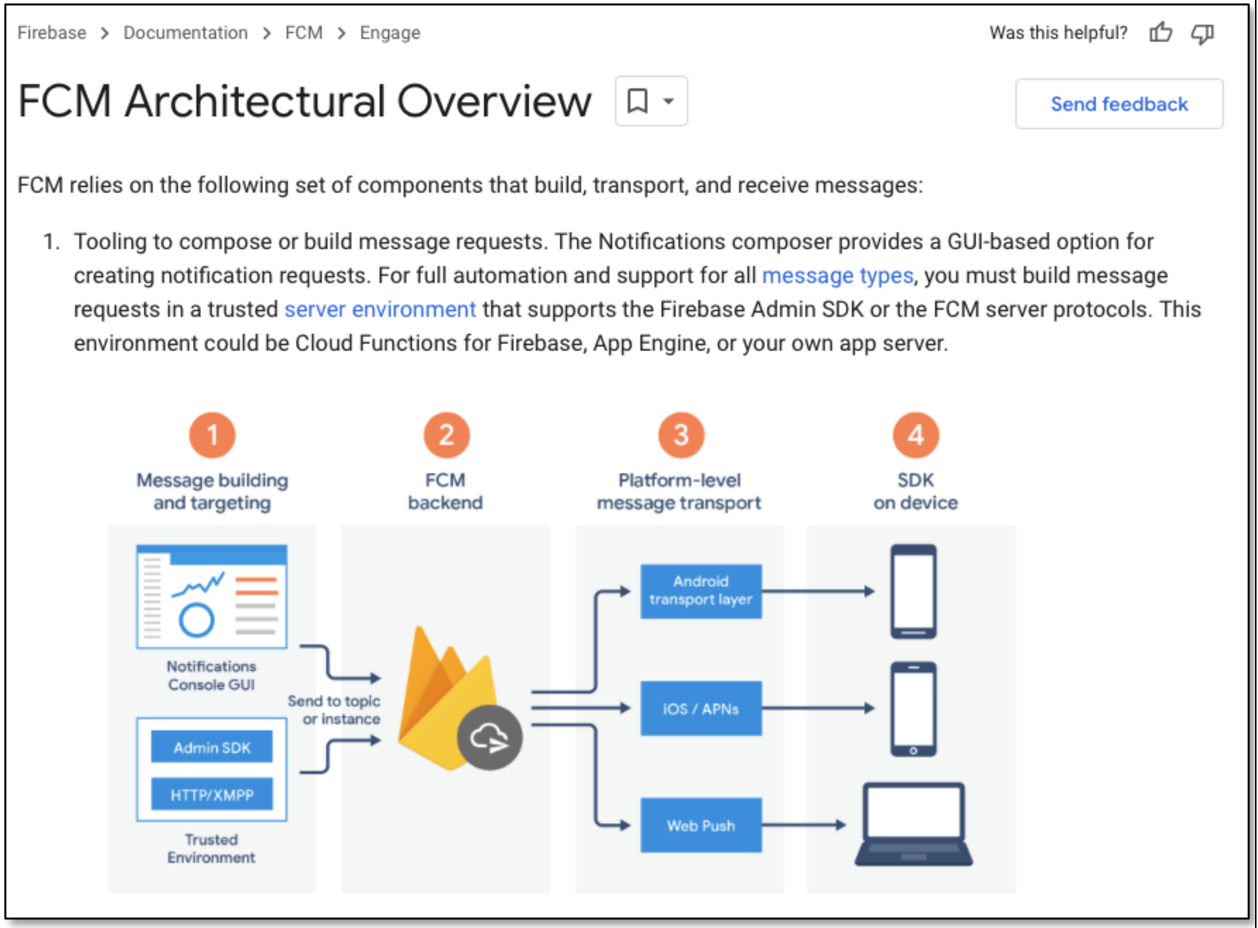
Issued Claim(s)	Public Documentation
1[pre] An end-user device comprising:	Samsung Galaxy phones and tablets and Samsung Tizen based devices are each "an end-user device." For example, the Galaxy S22 is a mobile device. *See e.g.*, https://www.samsung.com/us/smartphones/galaxy-s22/; As another example, Samsung's Tizen based devices are user devices. *See e.g.*,

	 https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/.
1[a] a modem for enabling communication with a network system over a service control link provided by the network system over a wireless access network, the service control link secured by an encryption protocol and configured to support control-plane communications between the network system and a service control device link agent on the end-user device;	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise "a modem for enabling communication with a network system over a service control link provided by the network system over a wireless access network, the service control link secured by an encryption protocol and configured to support control-plane communications between the network system and a service control device link agent on the end-user device." For example, the Galaxy S22 includes a plurality of wireless modems for communicating with a network system over a wi-fi network and/or a cellular network (a "wireless access network"). *See e.g.*,

	 Network & Connectivity 5G 5G Non-Standalone (NSA), Standalone (SA), Sub6 / mmWave LTE Enhanced 4x4 MIMO, Up to 7CA, LTE Cat.20 Up to 2.0Gbps Download / Up to 200Mbps Upload Wi-Fi Wi-Fi 802.11 a/b/g/n/ac/ax 2.4G+5GHz+6GHz, HE160, MIMO, 1024-QAM Up to 2.4Gbps Download / Up to 2.4Gbps Upload Bluetooth Bluetooth® v 5.2, USB type-C, NFC, Location(GPS, Galileo, Glonass, BeiDou) Ultra Wide Band *Requires optimal connection. Actual speed may vary depending on country, carrier and user environment. *The bandwidths supported by the device may vary depending on the region or service provider. *Download and upload speeds reaching up to 2.4Gbps only available with Wi-Fi 6E. Wi-Fi 6E only supported on Galaxy S22 Ultra and S22+. Galaxy S22 has Wi-Fi 6. *Galileo and BeiDou coverage may be limited. BeiDou may not be available for certain countries. https://www.samsung.com/us/smartphones/galaxy-s22/models/. Samsung Galaxy phones and tablets use encryption protocols to secure communications between the device and the network system. *See e.g.*,

	 SECURE Chip-up manufacturing Samsung designs, creates and validates every component we put in our mobile devices in highly secure factories, ensuring a consistent supply chain and end product you can trust. Data isolation Data isolation is built into every component of our mobile devices. It's a double-lock for sensitive data, meaning the only person who ever has access is you. Data encryption Knox platform encryption protects your data from vulnerabilities, even in the event of theft or loss. Even while your device is powered off, your data remains encrypted and won't be decrypted unless you choose to. Run-time protection Run-time protection protects your device against data attacks or malware. Any unauthorized or unintended attempts to access or modify your phone's core are blocked in real time.

	 12:45 Secure Wi-Fi Protected Secure Wi-Fi is protection of your security and privacy on this Wi-Fi network. Your plan 200/250 MB left Stop Auto protect Wi-Fi Automatically protect your connection when you connect to public or unsecured Wi-Fi. Protection plan Protected apps App protection history **Secure your connections** Secure Wi-Fi encrypts outgoing internet traffic and disables tracking apps and websites when using public wireless connections, allowing you to browse the internet safely without fear of security breaches. **Secure Wi-Fi feature availability may vary depending on country, carrier, or network environment and may not be supported on all Samsung mobile devices. Fees may apply depending on Secure Wi-Fi usage.* HOW TO SECURE YOUR WI-FI https://www.samsung.com/us/security/;



Firebase > Documentation > FCM > Engage

Was this helpful?

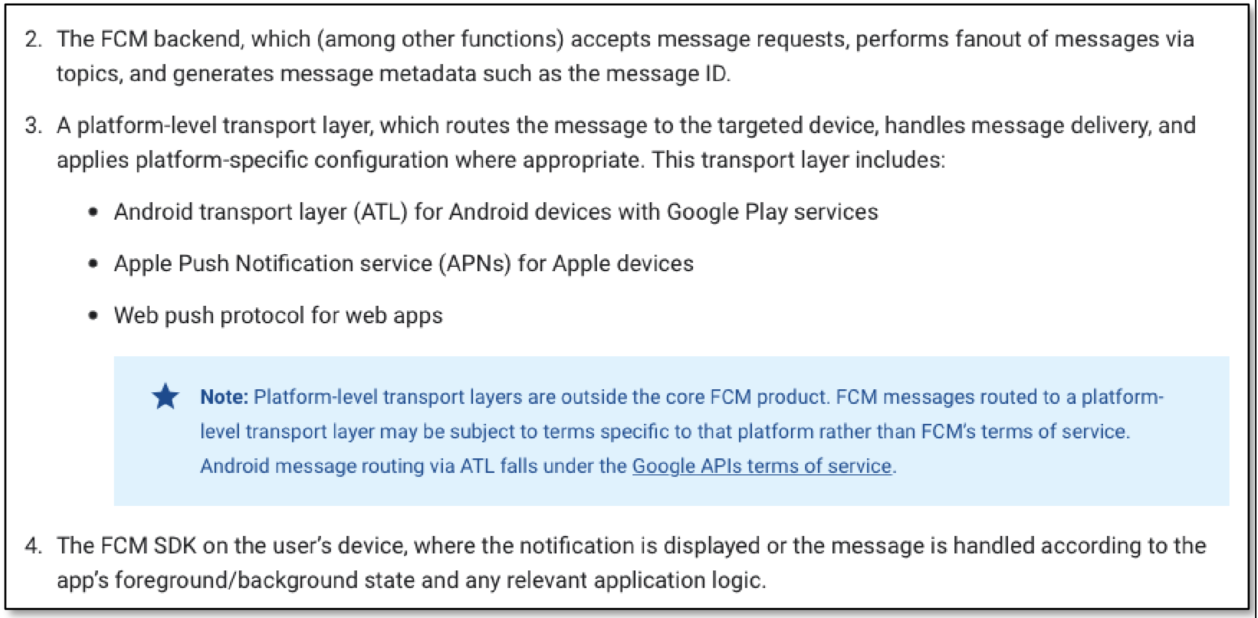
FCM Architectural Overview

Send feedback

FCM relies on the following set of components that build, transport, and receive messages:

1. Tooling to compose or build message requests. The Notifications composer provides a GUI-based option for creating notification requests. For full automation and support for all message types, you must build message requests in a trusted server environment that supports the Firebase Admin SDK or the FCM server protocols. This environment could be Cloud Functions for Firebase, App Engine, or your own app server.

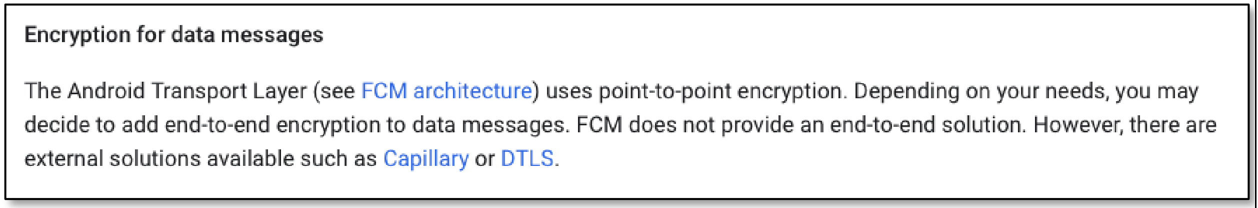
1
Message building and targeting
2
FCM backend
3
Platform-level message transport
4
SDK on device
Notifications Console GUI
Send to topic or instance
Admin SDK
HTTP/XMPP
Trusted Environment
Android transport layer
iOS / APNs
Web Push



2. The FCM backend, which (among other functions) accepts message requests, performs fanout of messages via topics, and generates message metadata such as the message ID.
3. A platform-level transport layer, which routes the message to the targeted device, handles message delivery, and applies platform-specific configuration where appropriate. This transport layer includes:
   - Android transport layer (ATL) for Android devices with Google Play services
   - Apple Push Notification service (APNs) for Apple devices
   - Web push protocol for web apps

   ★ **Note:** Platform-level transport layers are outside the core FCM product. FCM messages routed to a platform-level transport layer may be subject to terms specific to that platform rather than FCM's terms of service. Android message routing via ATL falls under the Google APIs terms of service.
4. The FCM SDK on the user's device, where the notification is displayed or the message is handled according to the app's foreground/background state and any relevant application logic.

https://firebase.google.com/docs/cloud-messaging/fcm-architecture;

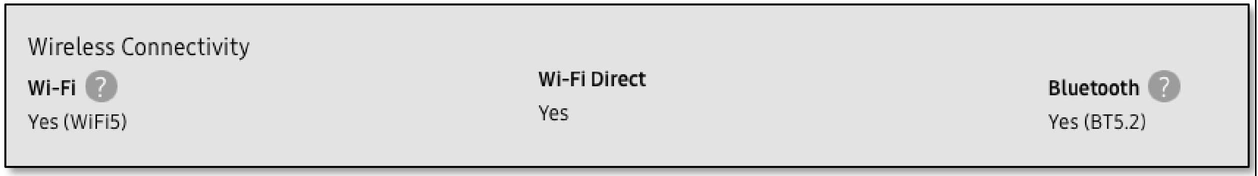
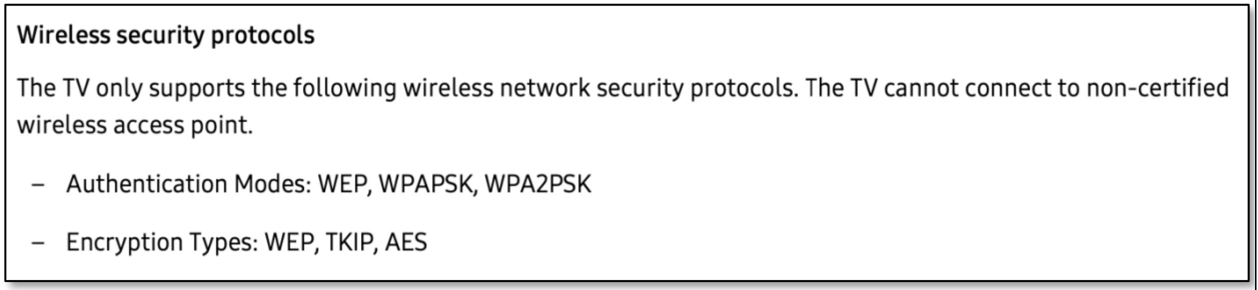


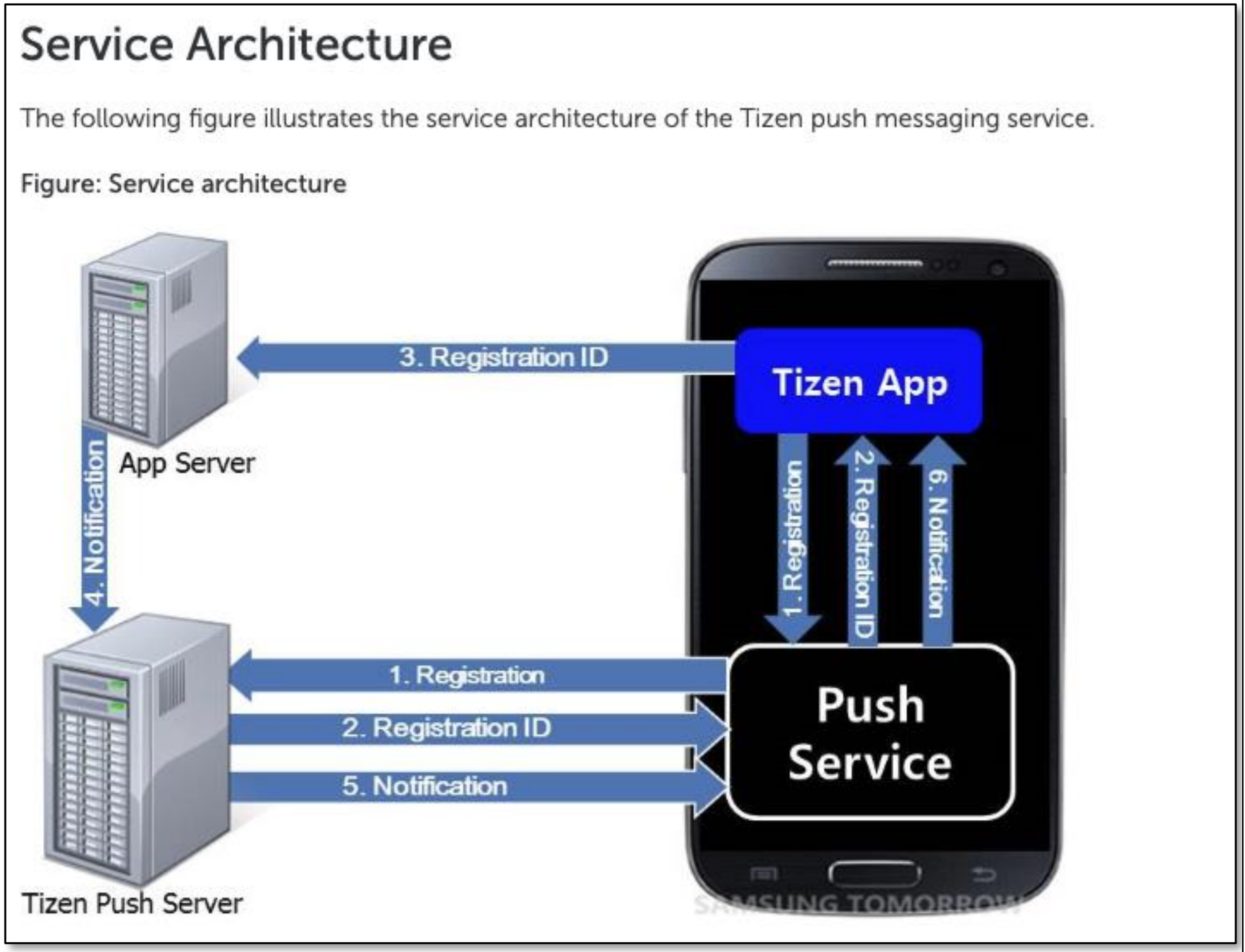
Encryption for data messages

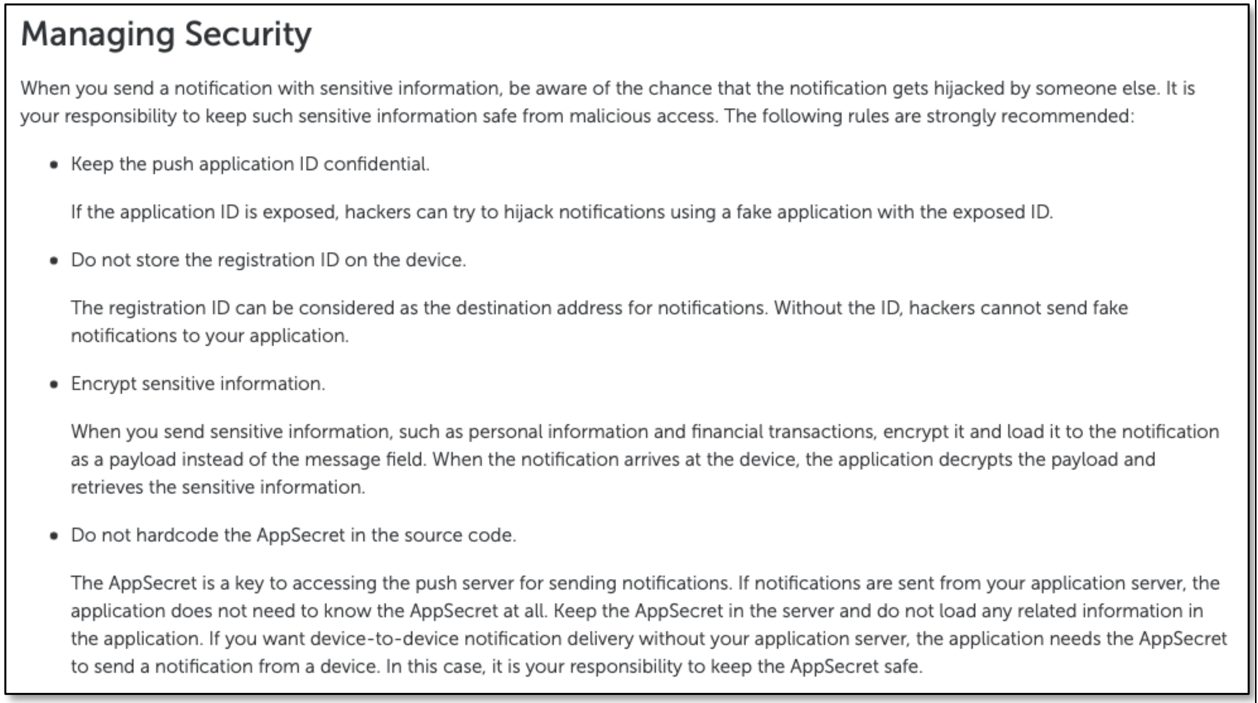
The Android Transport Layer (see FCM architecture) uses point-to-point encryption. Depending on your needs, you may decide to add end-to-end encryption to data messages. FCM does not provide an end-to-end solution. However, there are external solutions available such as Capillary or DTLS.

https://firebase.google.com/docs/cloud-messaging/concept-options#encryption_for_data_messages.

As another example, Samsung's Tizen based devices include wireless modems for communicating with a network system over a wi-fi network (a "wireless access network"). *See e.g.*,

	 Wireless Connectivity **Wi-Fi** Yes (WiFi5) **Wi-Fi Direct** Yes **Bluetooth** Yes (BT5.2) https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/#specs;  **Wireless security protocols** The TV only supports the following wireless network security protocols. The TV cannot connect to non-certified wireless access point. – Authentication Modes: WEP, WPAPSK, WPA2PSK – Encryption Types: WEP, TKIP, AES https://downloadcenter.samsung.com/content/UM/202203/20220303092001587/OSNATSCB-3.2.0_EM_Oscar_Pontus_Nike_Kant-SU2e_USA_ENG_220216.0.pdf. Further, Samsung's Tizen based devices use encryption protocols to secure communications between the device and the network system. *See e.g.*,

	 **Service Architecture** The following figure illustrates the service architecture of the Tizen push messaging service. Figure: Service architecture App Server 3. Registration ID Tizen App 4. Notification 1. Registration 2. Registration ID 6. Notification 1. Registration 2. Registration ID 5. Notification Push Service Tizen Push Server SAMSUNG TOMORROW



Managing Security

When you send a notification with sensitive information, be aware of the chance that the notification gets hijacked by someone else. It is your responsibility to keep such sensitive information safe from malicious access. The following rules are strongly recommended:

- Keep the push application ID confidential.

  If the application ID is exposed, hackers can try to hijack notifications using a fake application with the exposed ID.

- Do not store the registration ID on the device.

  The registration ID can be considered as the destination address for notifications. Without the ID, hackers cannot send fake notifications to your application.

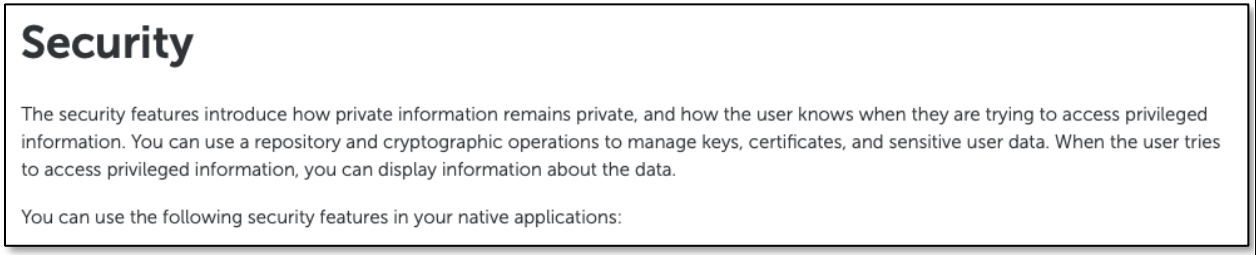
- Encrypt sensitive information.

  When you send sensitive information, such as personal information and financial transactions, encrypt it and load it to the notification as a payload instead of the message field. When the notification arrives at the device, the application decrypts the payload and retrieves the sensitive information.

- Do not hardcode the AppSecret in the source code.

  The AppSecret is a key to accessing the push server for sending notifications. If notifications are sent from your application server, the application does not need to know the AppSecret at all. Keep the AppSecret in the server and do not load any related information in the application. If you want device-to-device notification delivery without your application server, the application needs the AppSecret to send a notification from a device. In this case, it is your responsibility to keep the AppSecret safe.

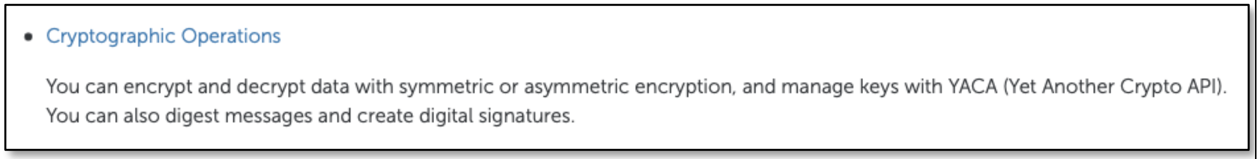
https://docs.tizen.org/application/native/guides/messaging/push/#connect;



Security

The security features introduce how private information remains private, and how the user knows when they are trying to access privileged information. You can use a repository and cryptographic operations to manage keys, certificates, and sensitive user data. When the user tries to access privileged information, you can display information about the data.

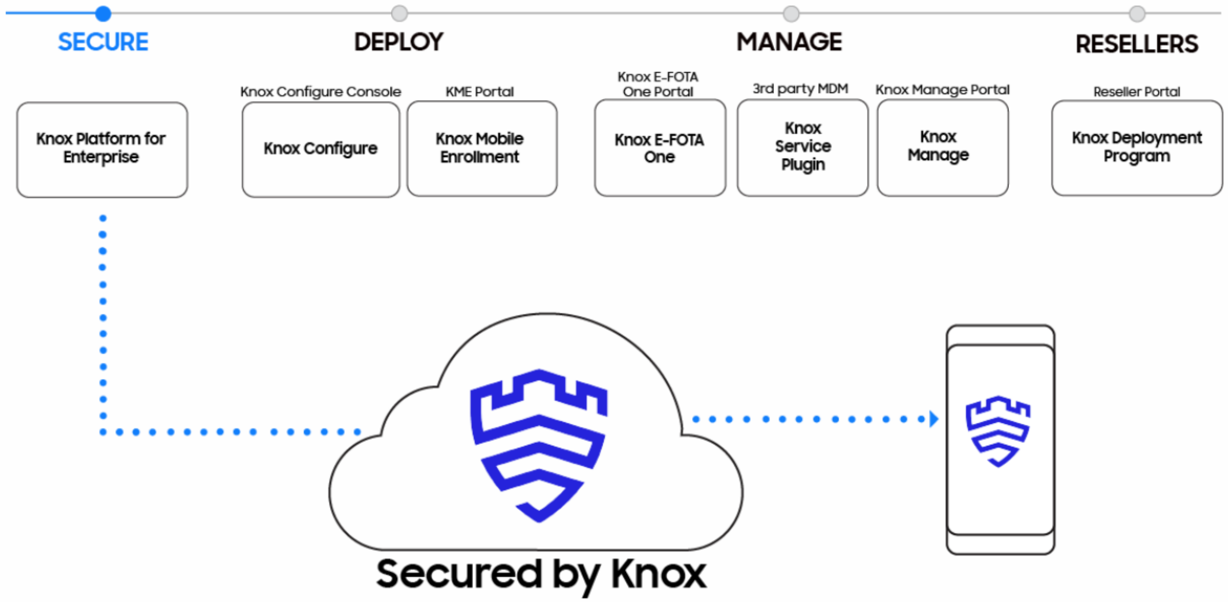
You can use the following security features in your native applications:

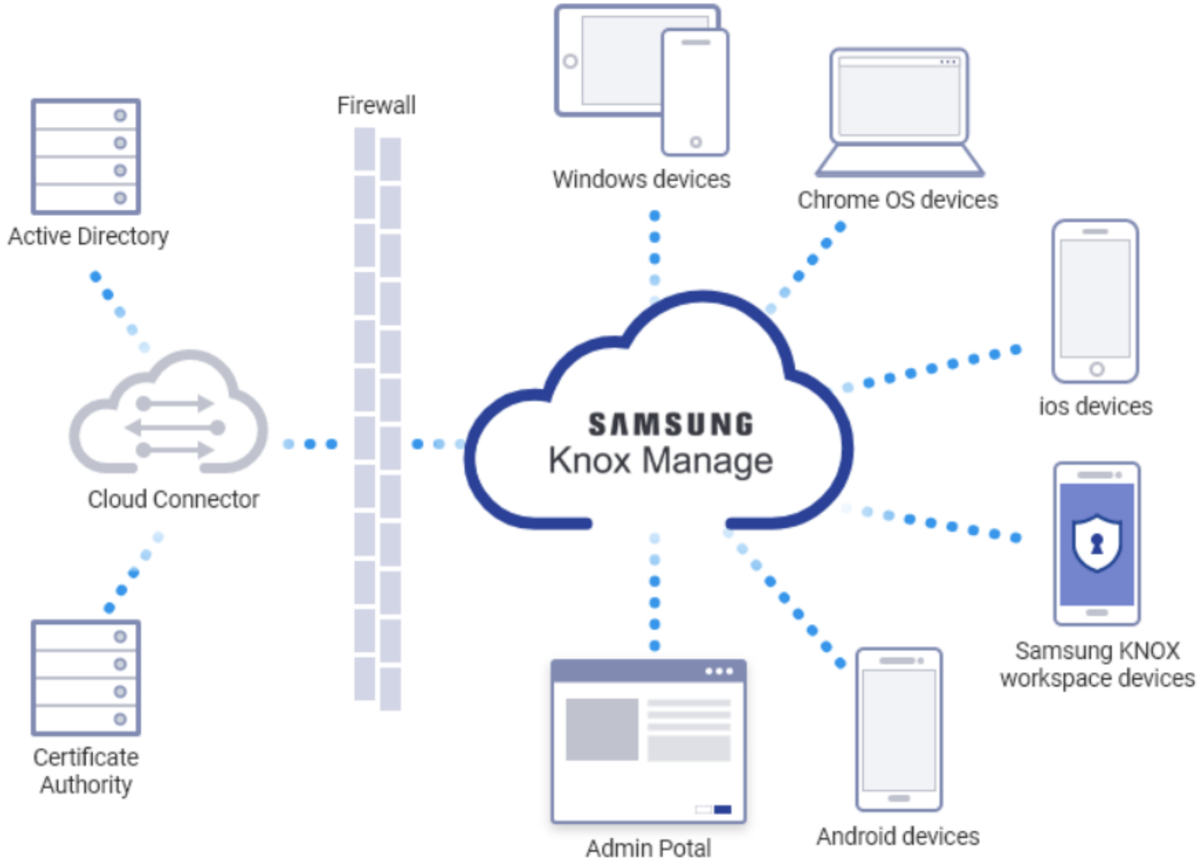


- Cryptographic Operations

  You can encrypt and decrypt data with symmetric or asymmetric encryption, and manage keys with YACA (Yet Another Crypto API). You can also digest messages and create digital signatures.

https://docs.tizen.org/application/native/guides/security/overview/.

	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, include modems for communicating with a network system over a service control link secured by encryption protocols programmed into the Samsung Knox MDM platform and e.g., Samsung Knox end-user device software for secure communications with a network system:  SECURE DEPLOY MANAGE RESELLERS Knox Platform for Enterprise Knox Configure Console Knox Configure KME Portal Knox Mobile Enrollment Knox E-FOTA One Portal Knox E-FOTA One 3rd party MDM Knox Service Plugin Knox Manage Portal Knox Manage Reseller Portal Knox Deployment Program Secured by Knox https://docs.samsungknox.com/admin/fundamentals/welcome.htm

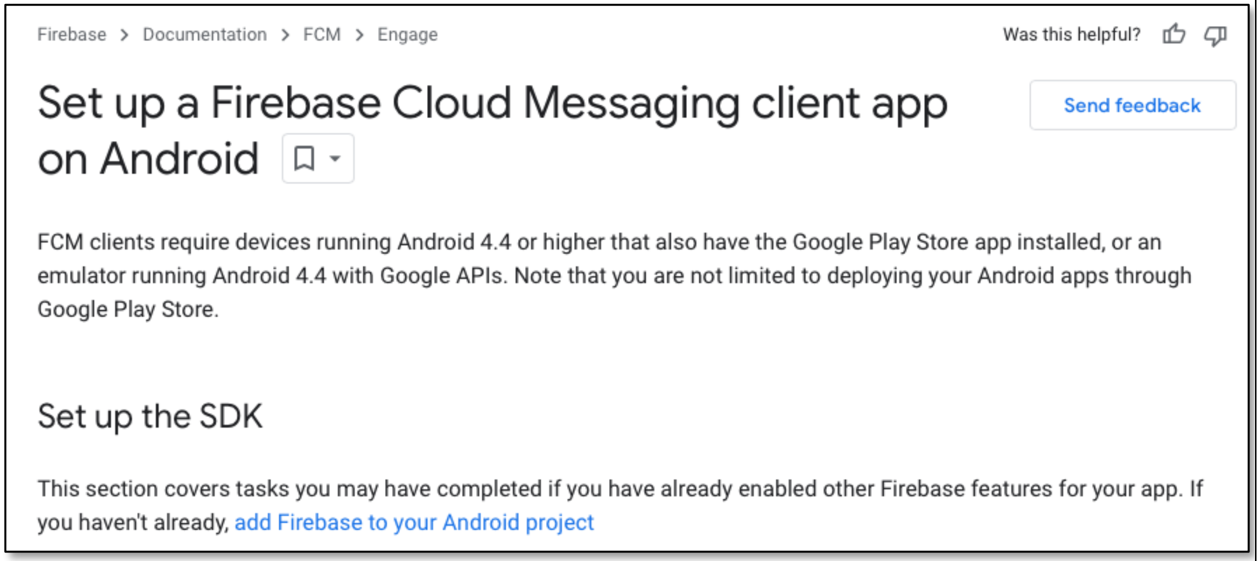


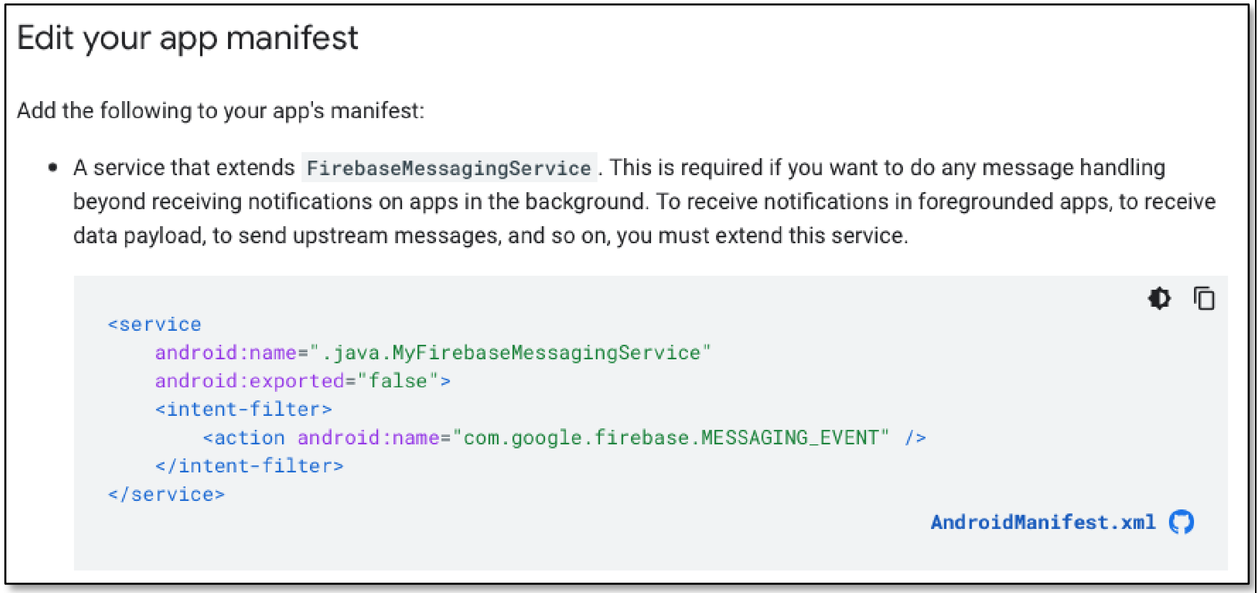
https://docs.samsungknox.com/admin/knox-manage/welcome.htm

As a further example, the communications link Samsung Knox Manage servers or Knox MDM platform and devices being managed by such servers or MDM platform, for example including the Knox workspace devices, Android devices, iOS devices, Chrome OS devices, and Windows devices illustrated above, comprise a "service control link." Data and messages transmitted over the secure control link between the Samsung Knox Manage servers or Knox MDM platform and managed devices is encrypted, and commands and notifications transmitted from such servers to devices are "agent messages."

As a further example, the communications link between Samsung's Tizen servers and Tizen OS devices, and Firebase messaging servers and Samsung's Android devices, comprises a "service

	control link." Notifications, data, and messages transmitted over these service control links from such servers to devices are encrypted, as illustrated above, and are "agent messages." As a further example, Samsung's Knox MDM platform, Knox Manage servers and Tizen servers are all examples of servers which provide control plane communications and which are service control server link elements, including because they control, manage, and apply service policies to user devices to which they are connected. As a further example, devices managed by Samsung Knox Manage or the Knox MDM platform are managed by software including specific end-user device applications and software, such as the Knox Manage (KM) agent on such devices (see, e.g., https://docs.samsungknox.com/admin/knox-manage/enroll-a-single-device.htm).

1[b] a plurality of device agents communicatively coupled to the service control device link agent through an agent communication bus, each of the plurality of device agents identifiable by an associated device agent identifier; and	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise "a plurality of device agents communicatively coupled to the service control device link agent through an agent communication bus, each of the plurality of device agents identifiable by an associated device agent identifier." For example, Samsung Galaxy phones and tablets comprise multiple device agents which have an identifier. *See e.g.*,  Firebase > Documentation > FCM > Engage Was this helpful? Send feedback **Set up a Firebase Cloud Messaging client app on Android** FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store. **Set up the SDK** This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project



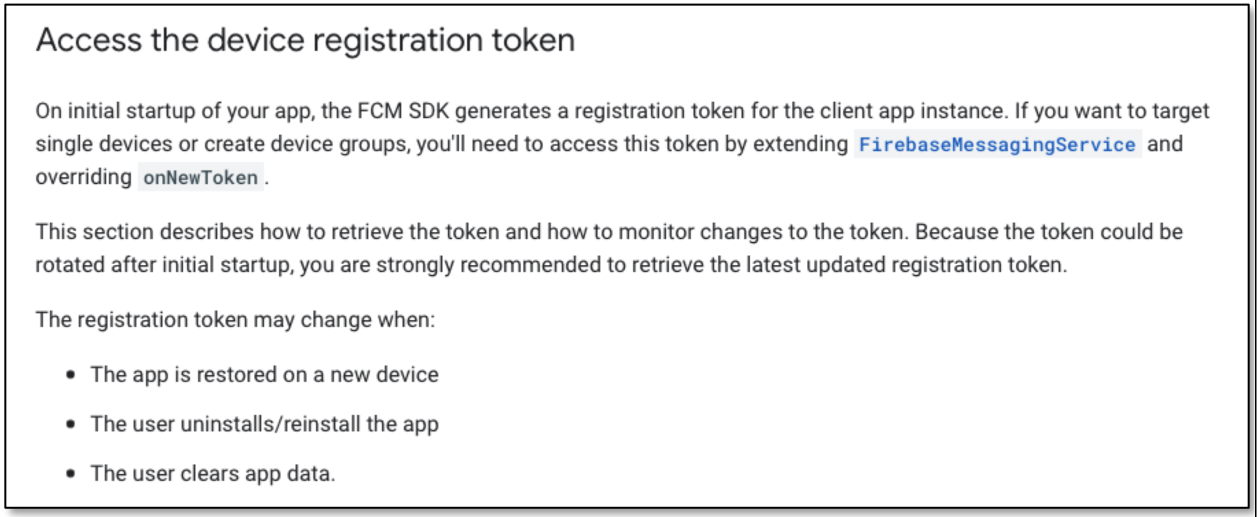
Edit your app manifest

Add the following to your app's manifest:

- A service that extends `FirebaseMessagingService`. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service.

```
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

AndroidManifest.xml



Access the device registration token

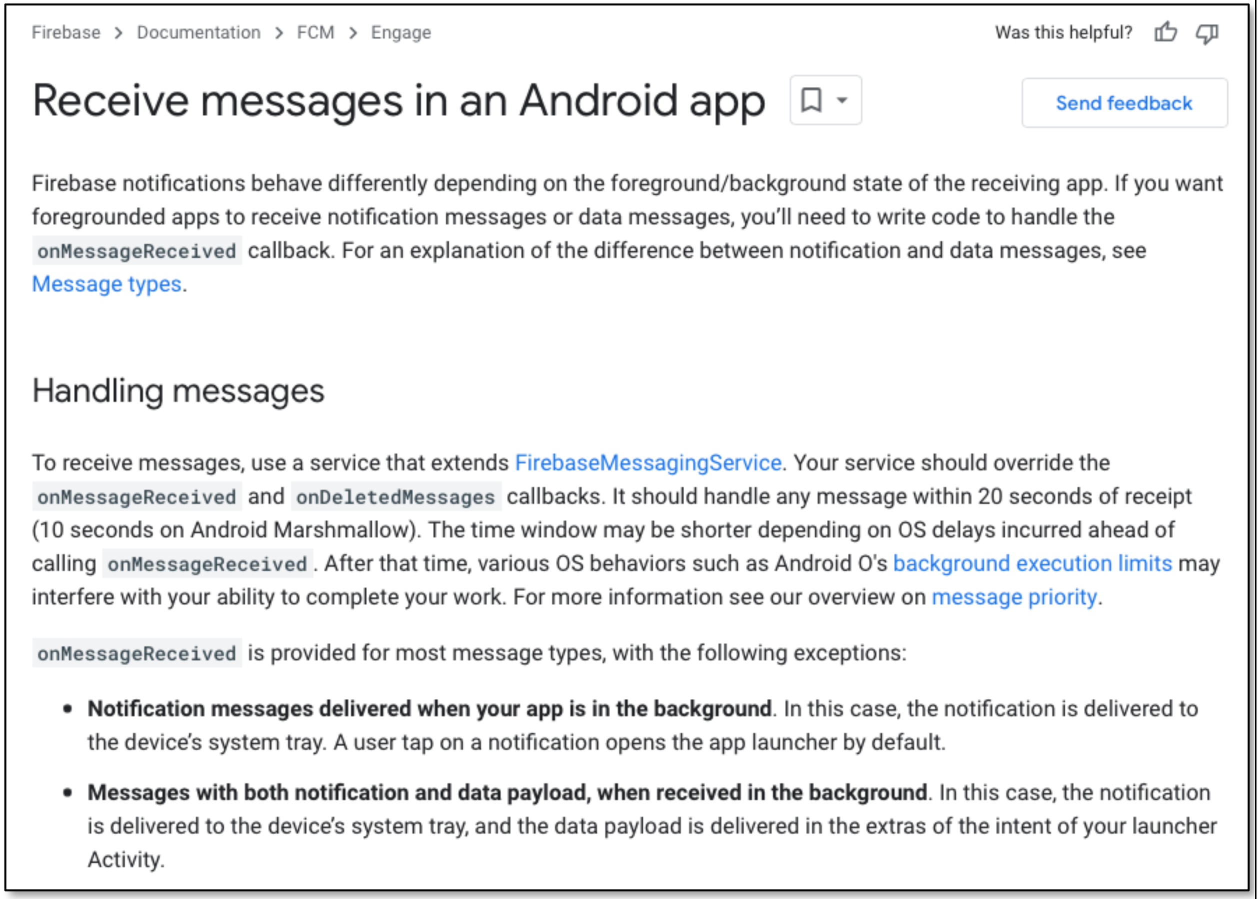
On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`.

This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

https://firebase.google.com/docs/cloud-messaging/android/client;



Receive messages in an Android app

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see Message types.

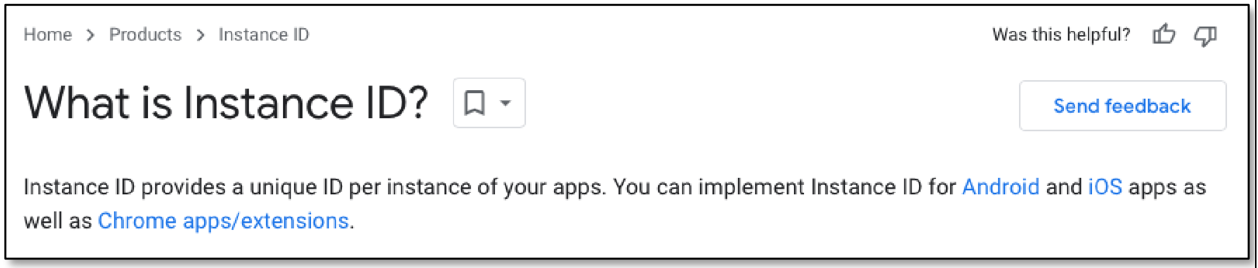
Handling messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's background execution limits may interfere with your ability to complete your work. For more information see our overview on message priority.

`onMessageReceived` is provided for most message types, with the following exceptions:

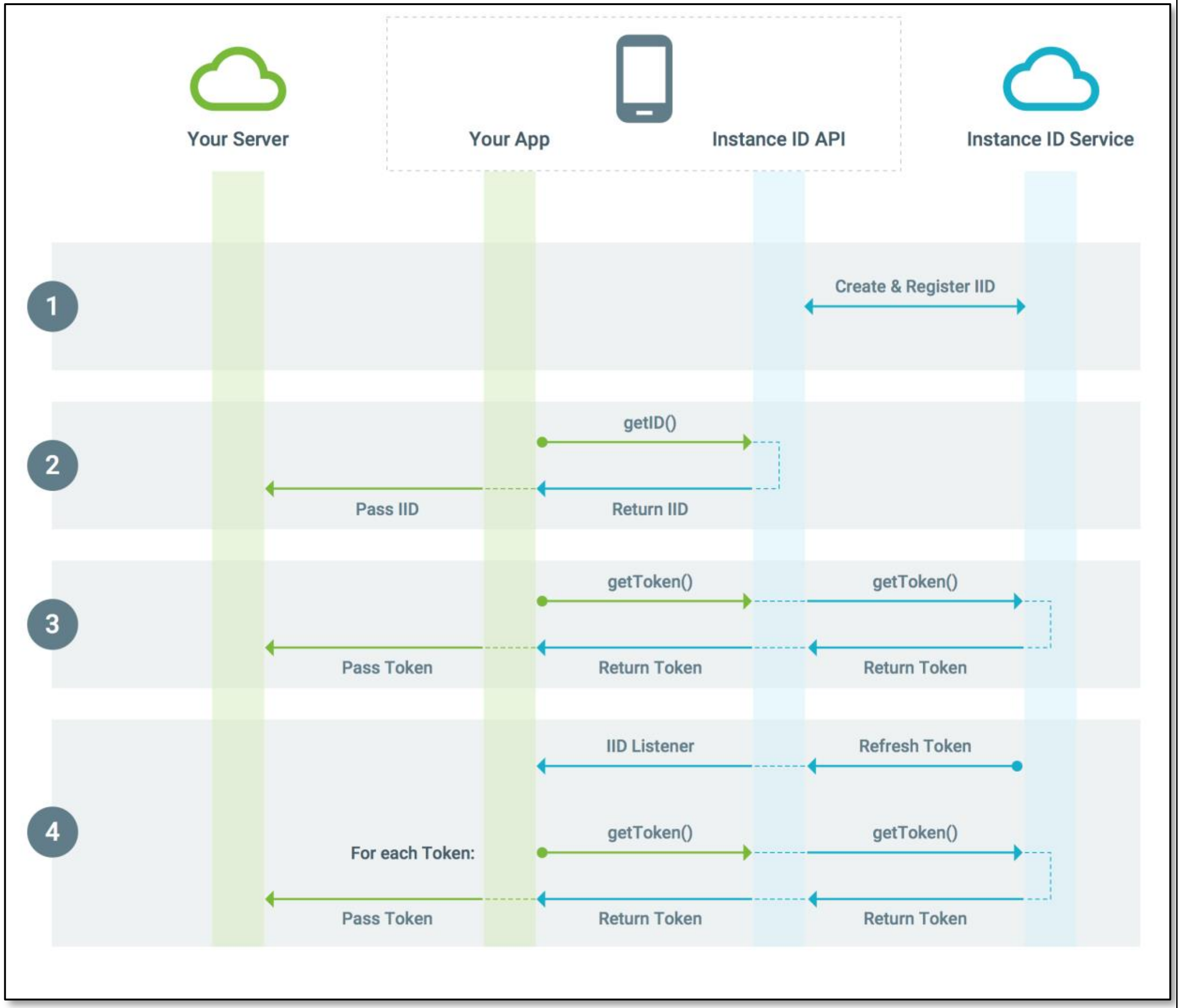
- **Notification messages delivered when your app is in the background**. In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

https://firebase.google.com/docs/cloud-messaging/android/receive;



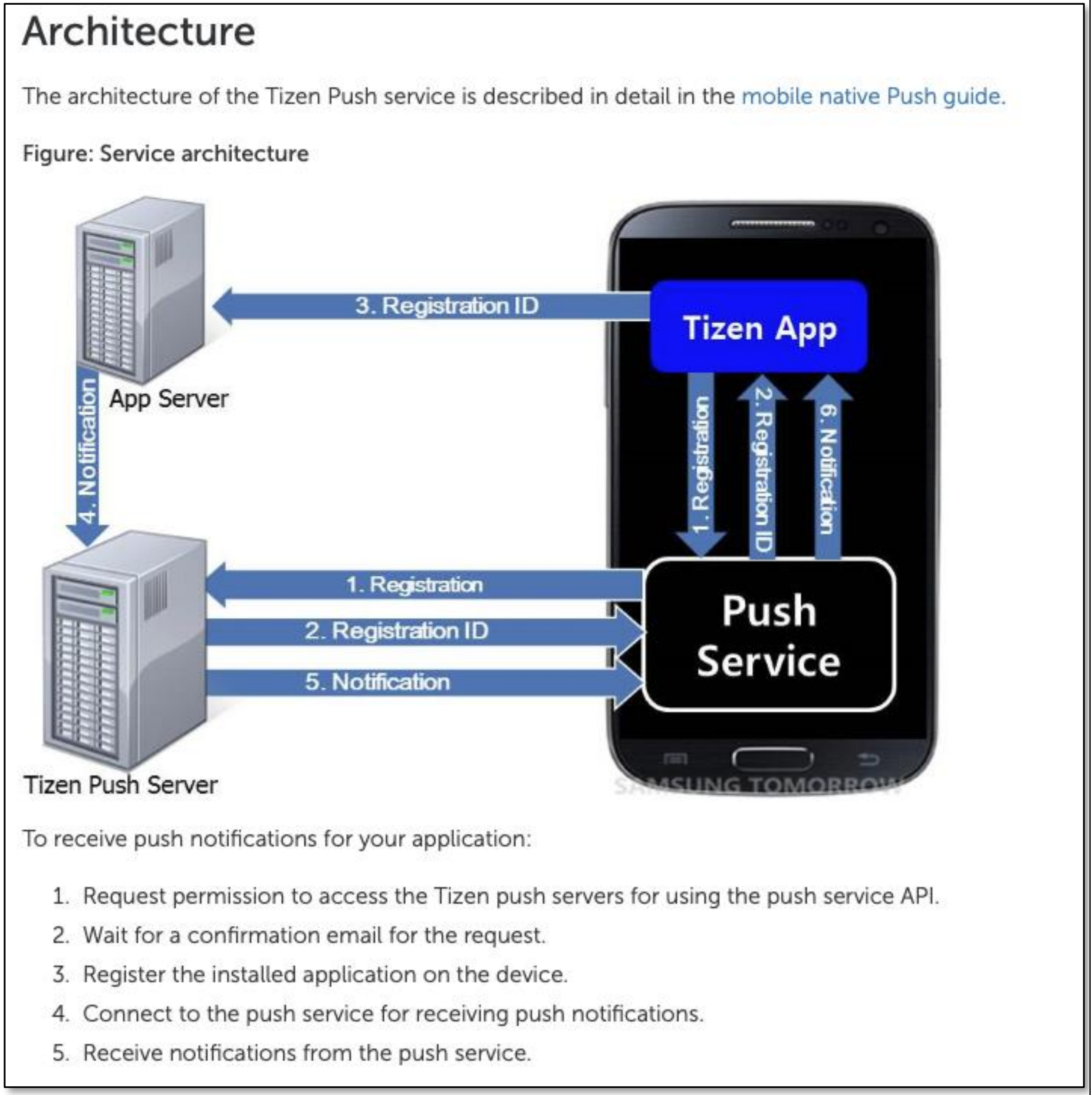
What is Instance ID?

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for Android and iOS apps as well as Chrome apps/extensions.



https://developers.google.com/instance-id.

For further example, Samsung's Tizen based devices comprise multiple device agents which have an identifier. *See e.g.*,



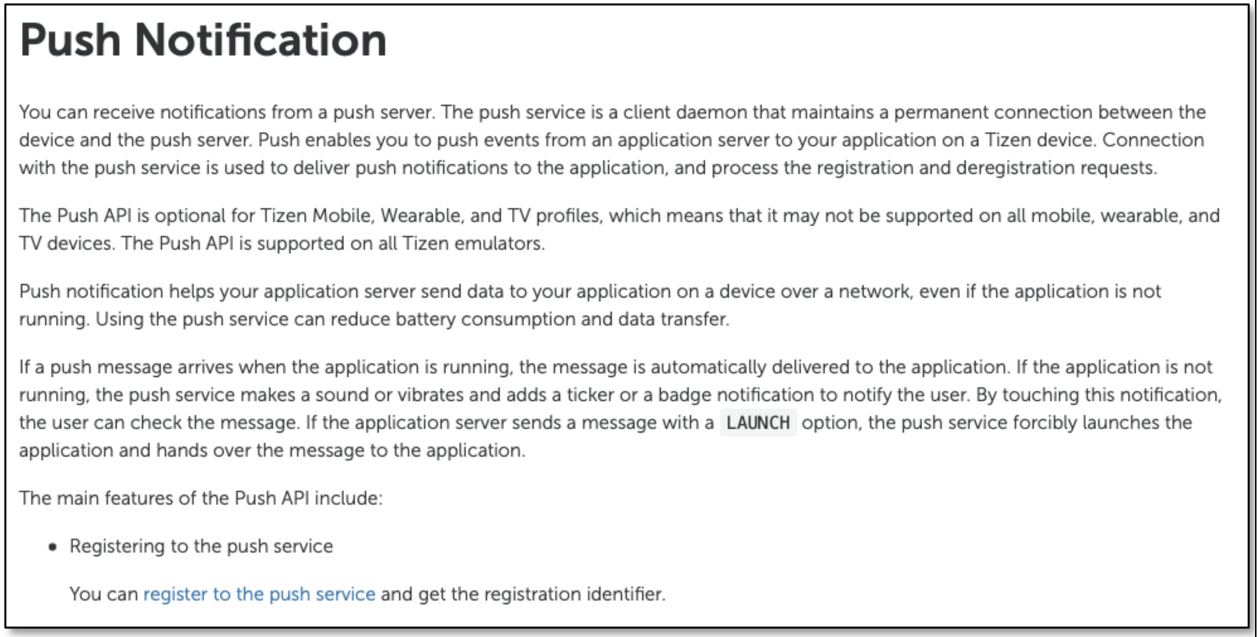
Architecture

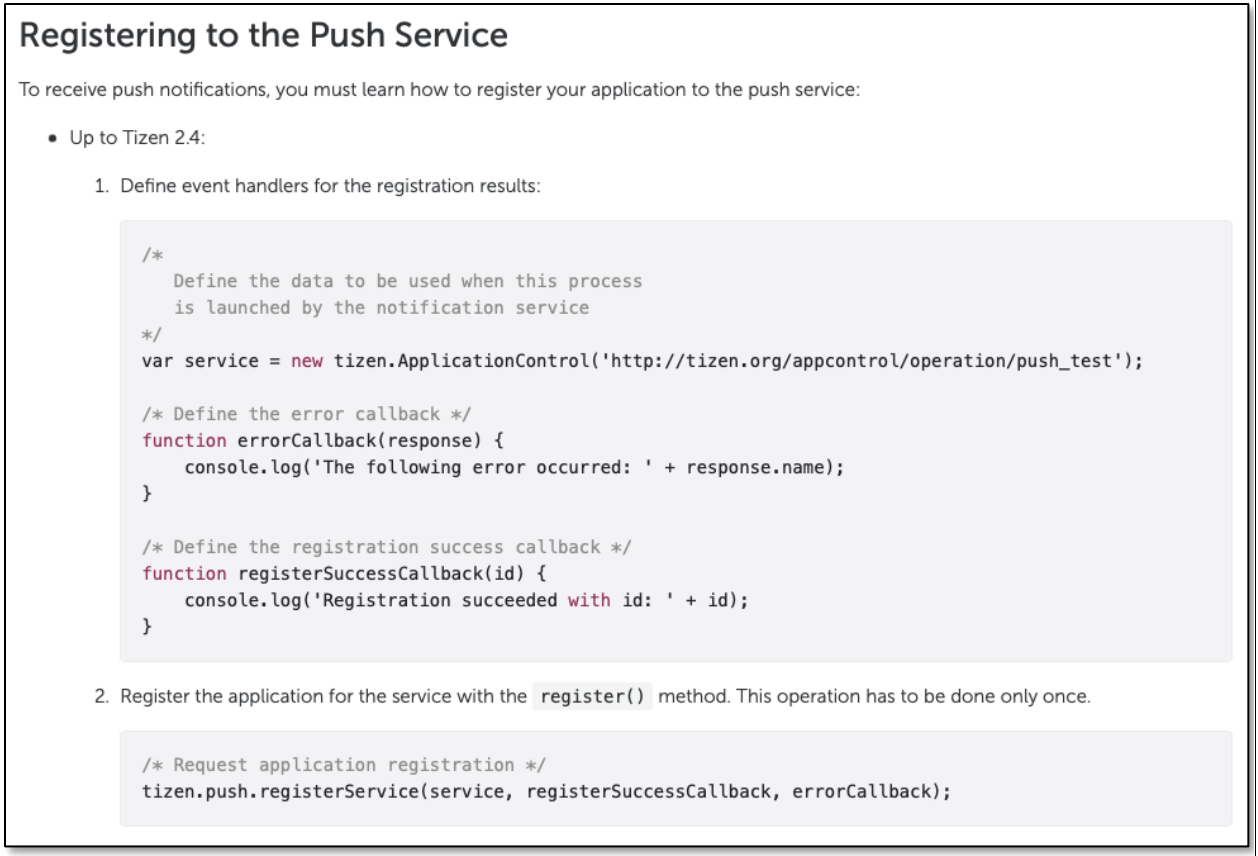
The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture

To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

	 **Push Notification** You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests. The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators. Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application. The main features of the Push API include: - Registering to the push service You can register to the push service and get the registration identifier.



Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

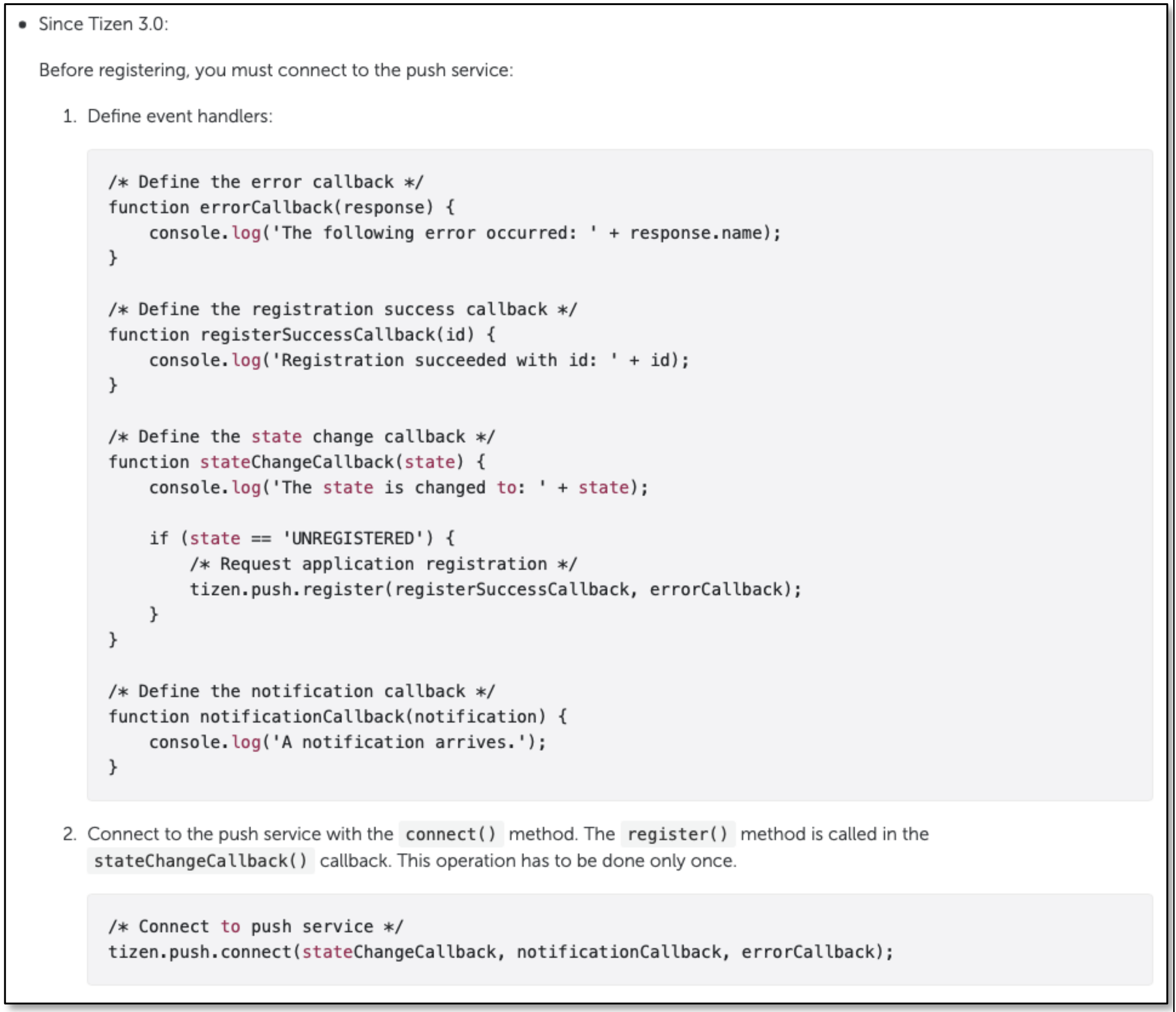
- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

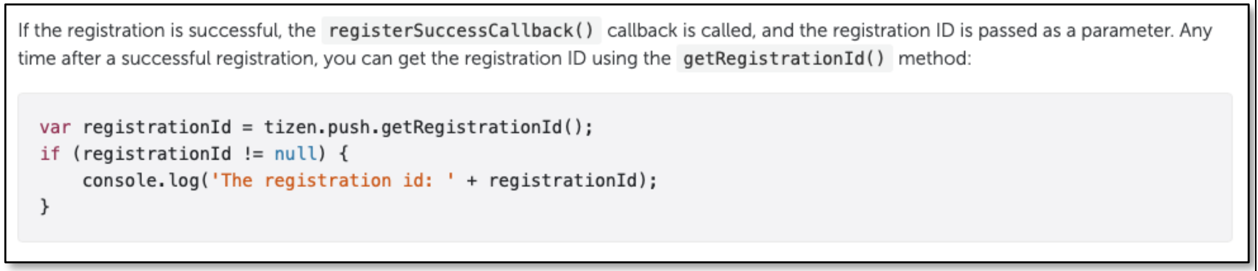
  Before registering, you must connect to the push service:

  1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

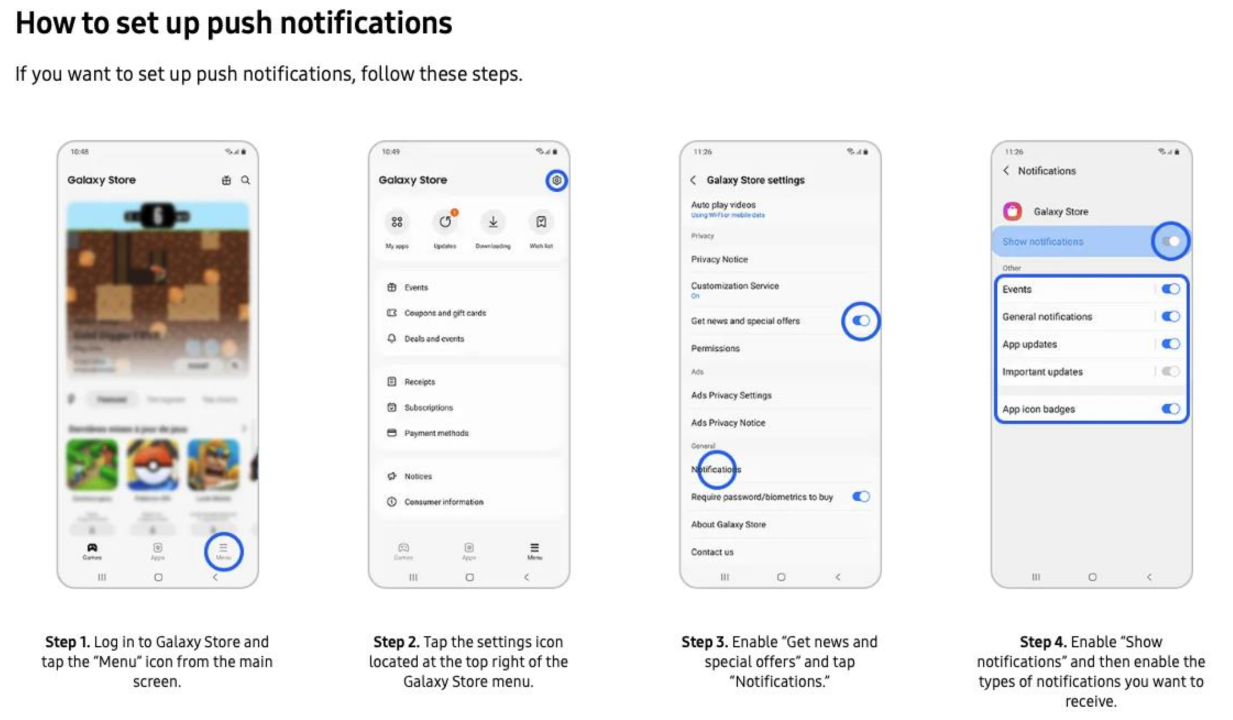
```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```



If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```

https://docs.tizen.org/application/web/guides/messaging/push/;



How to set up push notifications

If you want to set up push notifications, follow these steps.

Step 1. Log in to Galaxy Store and tap the "Menu" icon from the main screen.

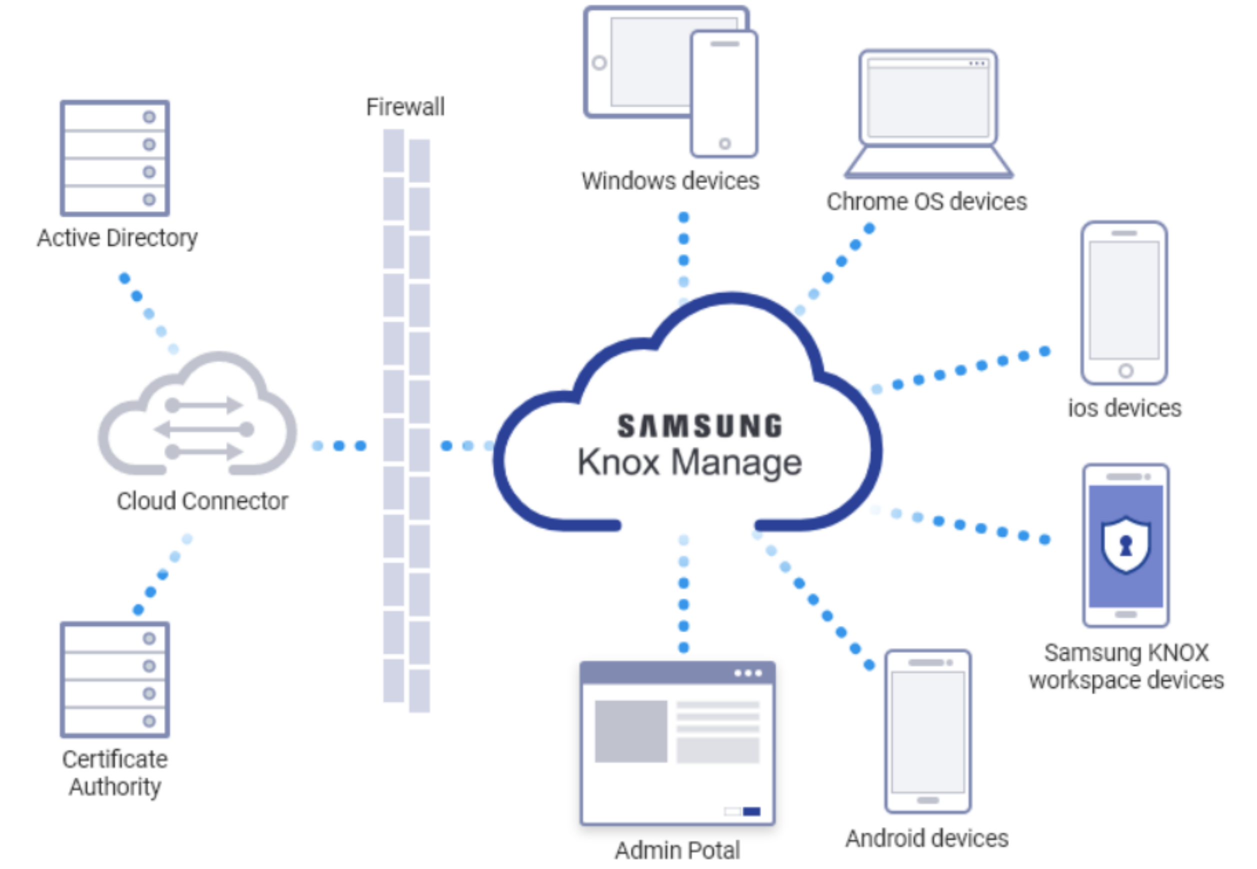
Step 2. Tap the settings icon located at the top right of the Galaxy Store menu.

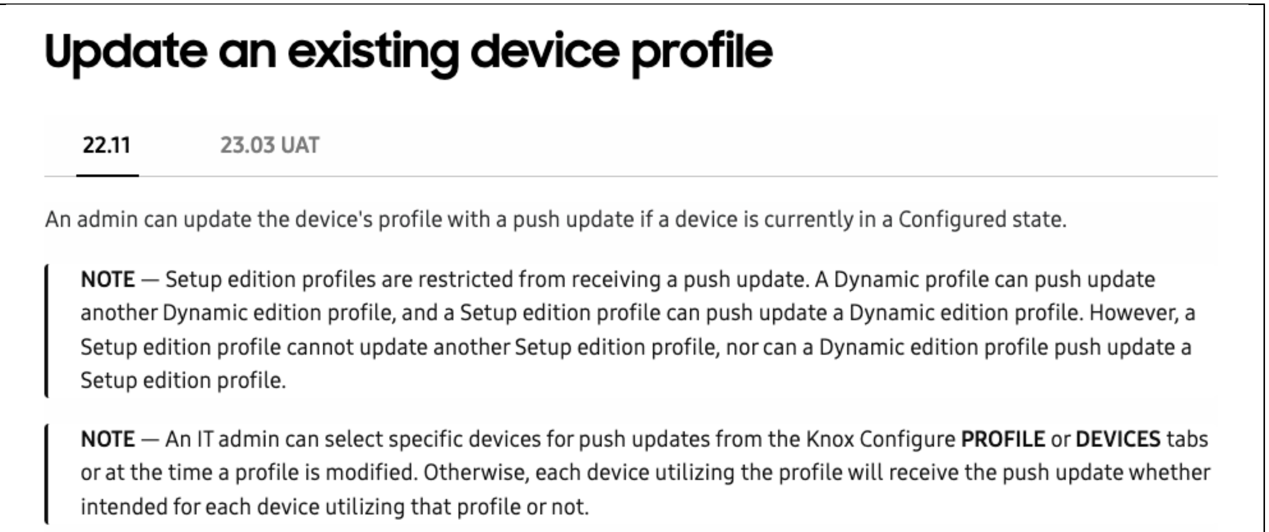
Step 3. Enable "Get news and special offers" and tap "Notifications."

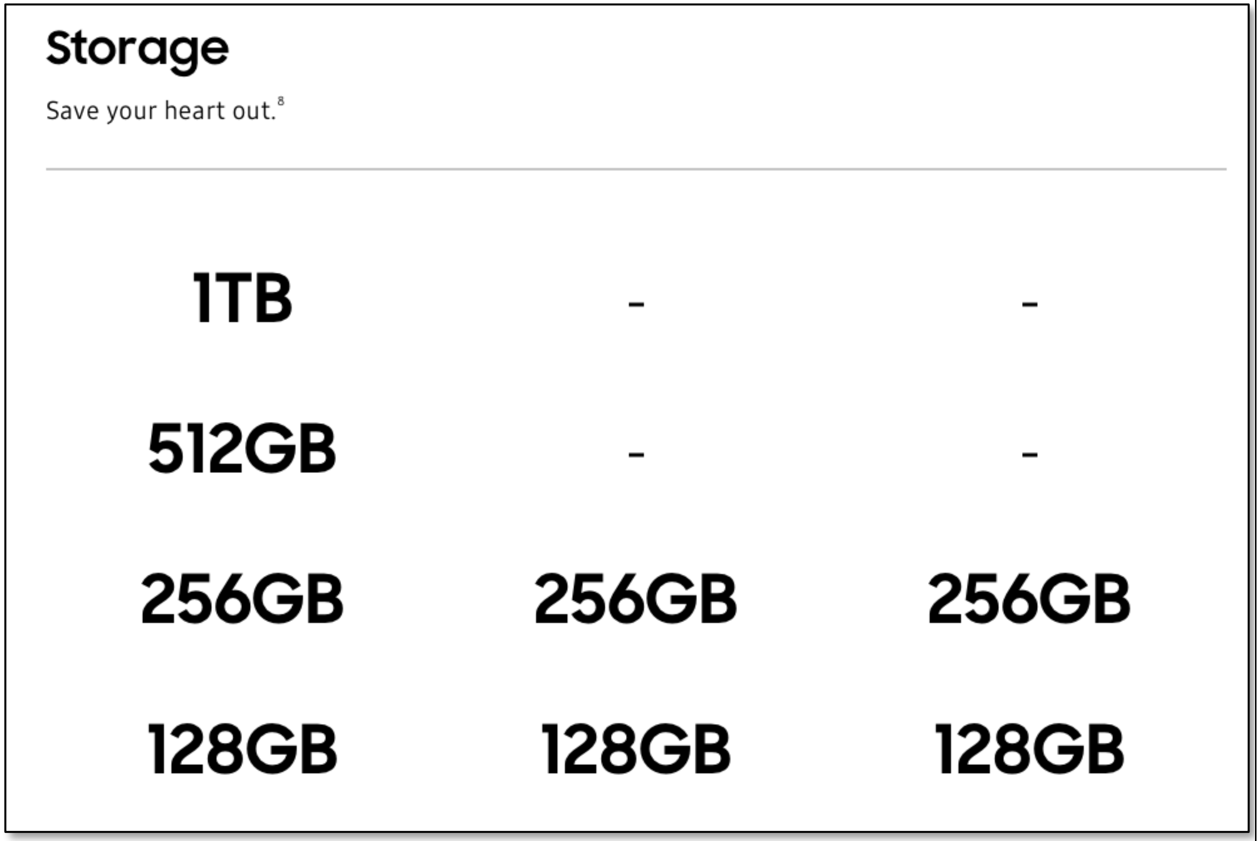
Step 4. Enable "Show notifications" and then enable the types of notifications you want to receive.

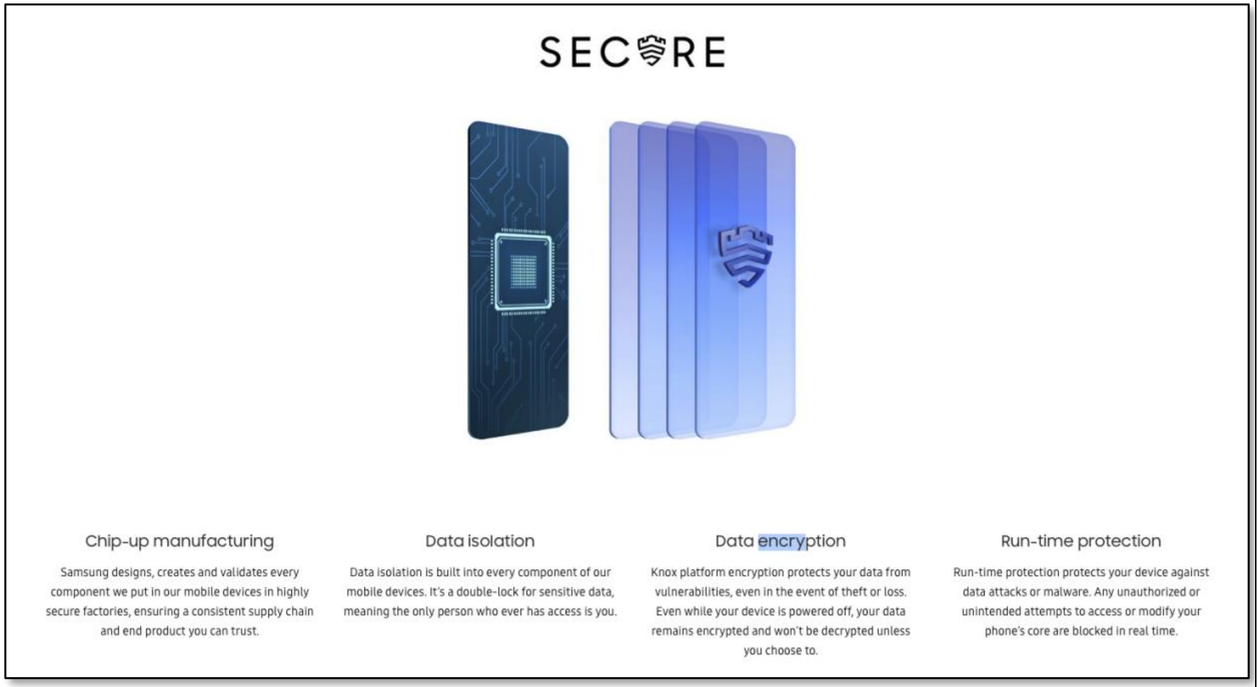
https://www.samsung.com/hk_en/support/apps-services/how-do-i-enable-push-notifications-from-galaxy-store/.

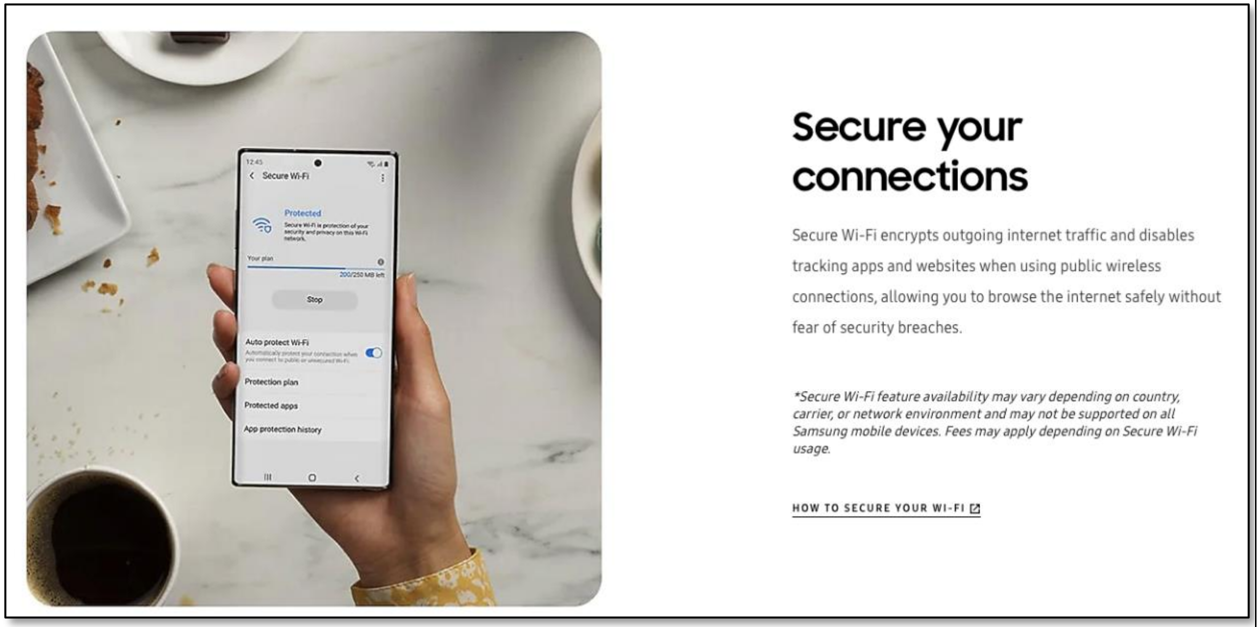
As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, comprise multiple device agents which have an identifier.

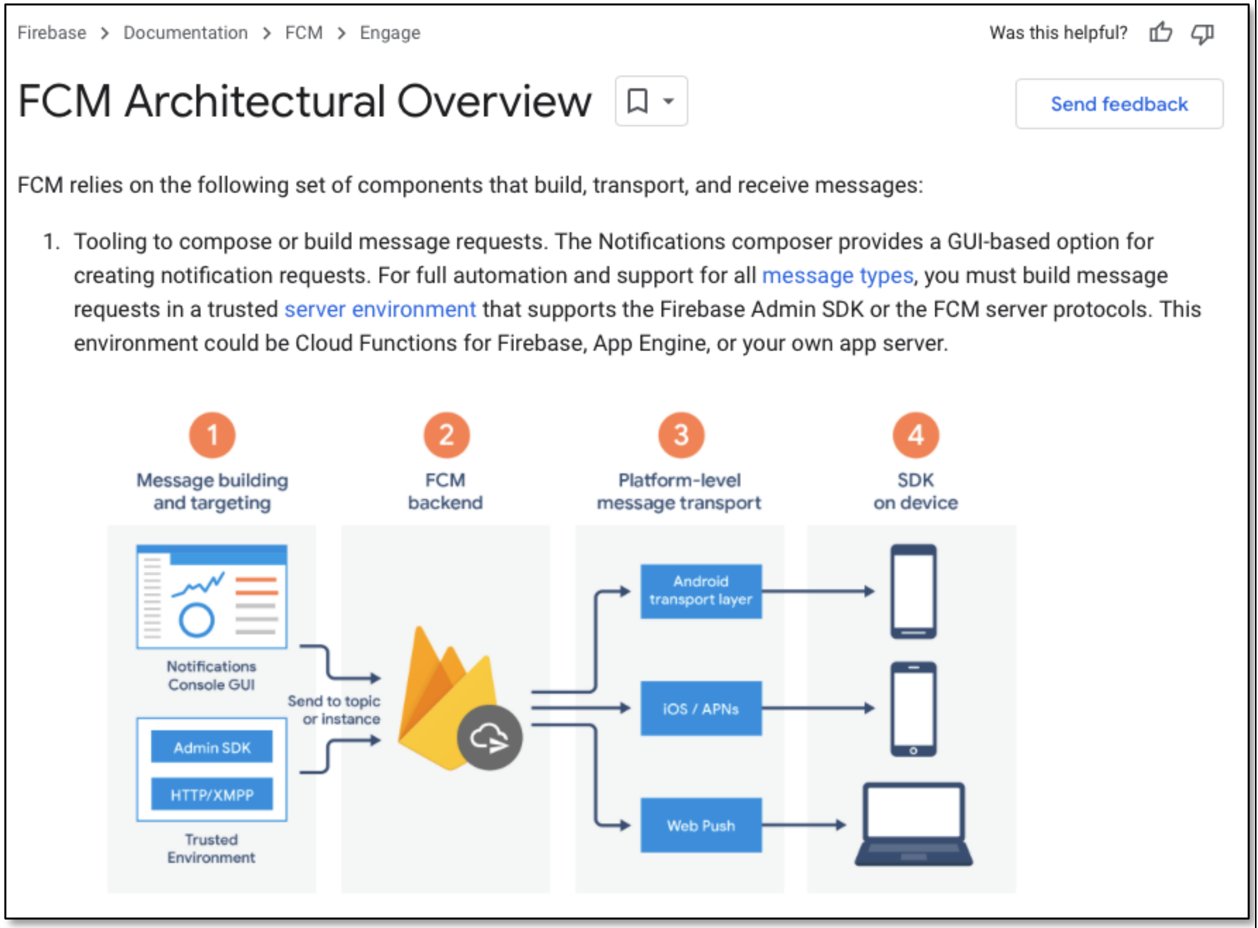
	 Active Directory Firewall Windows devices Chrome OS devices Cloud Connector SAMSUNG Knox Manage ios devices Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 **Update an existing device profile** 22.11 23.03 UAT An admin can update the device's profile with a push update if a device is currently in a Configured state. **NOTE** — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile. **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not. https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm As a further illustration, the service control device link agents on Samsung's devices are coupled to various other device agents within the device by way of the Android operating system through an agent communication bus. That communication structure allows for intercommunication between agents in the device, and enables the service control device link agents to send messages to (and control) behavior of other components within the system. The communications bus connecting device agents to the service control device link agents need not be internal to the device, and the communications channel between Samsung Knox and MDM servers and managed devices is an example of such a communications bus.
1[c] memory configured to store an encryption key, the encryption key shared between the service control device link agent and a service control server link element of the network system;	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise "memory configured to store an encryption key, the encryption key shared between the service control device link agent and a service control server link element of the network system." For example, Samsung Galaxy phones and tablets include memory that stores an encryption key that is shared between the agent and the network system. *See, e.g.*,

<table>
<tr>
<td></td>
<td>

<b>Storage</b>
Save your heart out.[8]
<table>
<tr><td>1TB</td><td>-</td><td>-</td></tr>
<tr><td>512GB</td><td>-</td><td>-</td></tr>
<tr><td>256GB</td><td>256GB</td><td>256GB</td></tr>
<tr><td>128GB</td><td>128GB</td><td>128GB</td></tr>
</table>
https://www.samsung.com/us/smartphones/galaxy-s22/models/;
</td>
</tr>
</table>

	 SECURE Chip-up manufacturing Samsung designs, creates and validates every component we put in our mobile devices in highly secure factories, ensuring a consistent supply chain and end product you can trust. Data isolation Data isolation is built into every component of our mobile devices. It's a double-lock for sensitive data, meaning the only person who ever has access is you. Data encryption Knox platform encryption protects your data from vulnerabilities, even in the event of theft or loss. Even while your device is powered off, your data remains encrypted and won't be decrypted unless you choose to. Run-time protection Run-time protection protects your device against data attacks or malware. Any unauthorized or unintended attempts to access or modify your phone's core are blocked in real time.

	 Secure Wi-Fi Protected Stop Auto protect Wi-Fi Protection plan Protected apps App protection history **Secure your connections** Secure Wi-Fi encrypts outgoing internet traffic and disables tracking apps and websites when using public wireless connections, allowing you to browse the internet safely without fear of security breaches. **Secure Wi-Fi feature availability may vary depending on country, carrier, or network environment and may not be supported on all Samsung mobile devices. Fees may apply depending on Secure Wi-Fi usage.* HOW TO SECURE YOUR WI-FI https://www.samsung.com/us/security/;



Firebase > Documentation > FCM > Engage

Was this helpful?

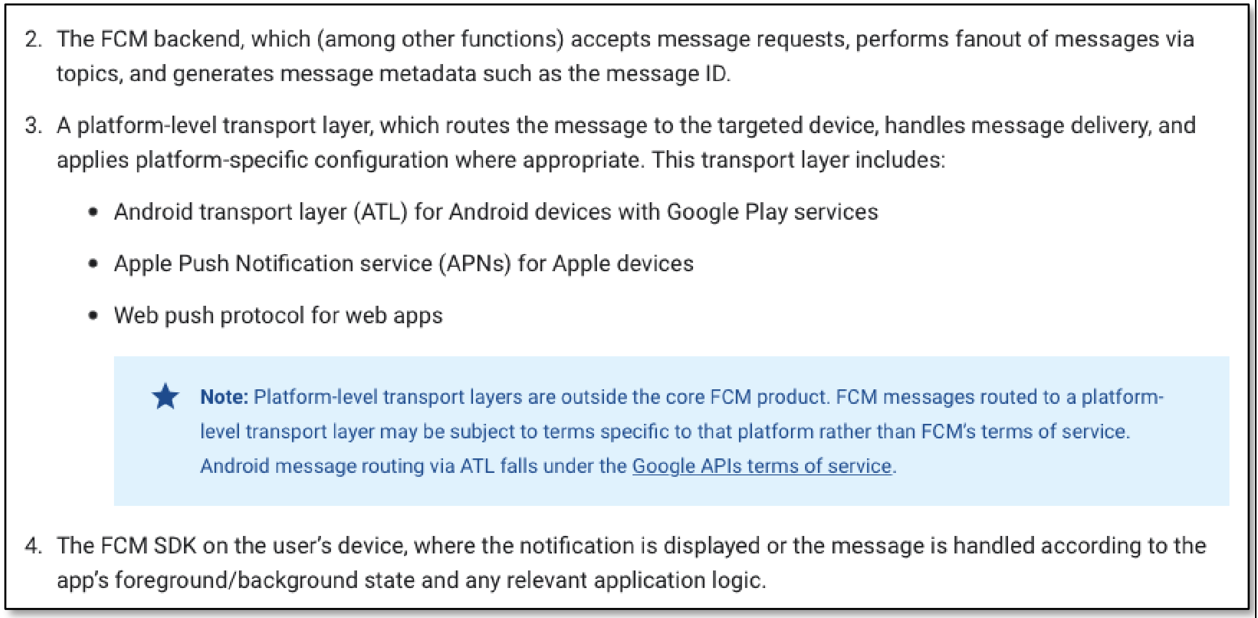
FCM Architectural Overview

Send feedback

FCM relies on the following set of components that build, transport, and receive messages:

1. Tooling to compose or build message requests. The Notifications composer provides a GUI-based option for creating notification requests. For full automation and support for all message types, you must build message requests in a trusted server environment that supports the Firebase Admin SDK or the FCM server protocols. This environment could be Cloud Functions for Firebase, App Engine, or your own app server.

1
Message building and targeting
2
FCM backend
3
Platform-level message transport
4
SDK on device
Notifications Console GUI
Send to topic or instance
Admin SDK
HTTP/XMPP
Trusted Environment
Android transport layer
iOS / APNs
Web Push

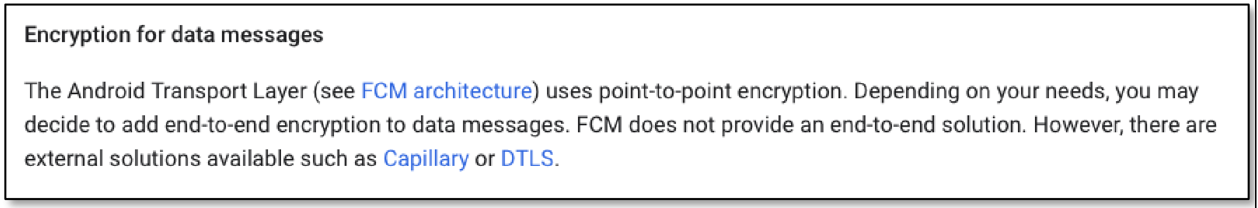


2. The FCM backend, which (among other functions) accepts message requests, performs fanout of messages via topics, and generates message metadata such as the message ID.
3. A platform-level transport layer, which routes the message to the targeted device, handles message delivery, and applies platform-specific configuration where appropriate. This transport layer includes:
   - Android transport layer (ATL) for Android devices with Google Play services
   - Apple Push Notification service (APNs) for Apple devices
   - Web push protocol for web apps

   ★ **Note:** Platform-level transport layers are outside the core FCM product. FCM messages routed to a platform-level transport layer may be subject to terms specific to that platform rather than FCM's terms of service. Android message routing via ATL falls under the Google APIs terms of service.

4. The FCM SDK on the user's device, where the notification is displayed or the message is handled according to the app's foreground/background state and any relevant application logic.

https://firebase.google.com/docs/cloud-messaging/fcm-architecture;

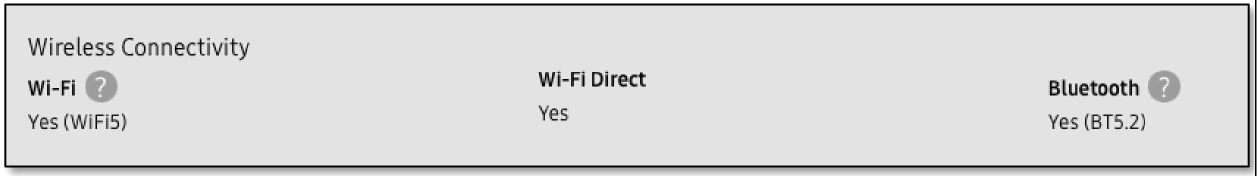


Encryption for data messages

The Android Transport Layer (see FCM architecture) uses point-to-point encryption. Depending on your needs, you may decide to add end-to-end encryption to data messages. FCM does not provide an end-to-end solution. However, there are external solutions available such as Capillary or DTLS.

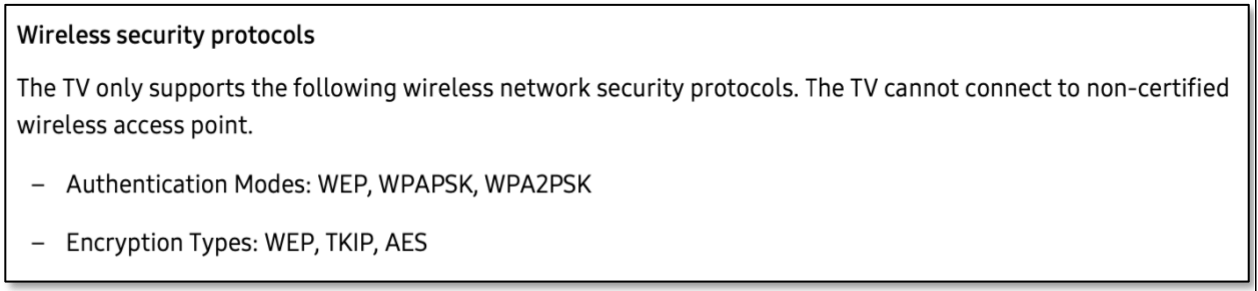
https://firebase.google.com/docs/cloud-messaging/concept-options#encryption_for_data_messages.

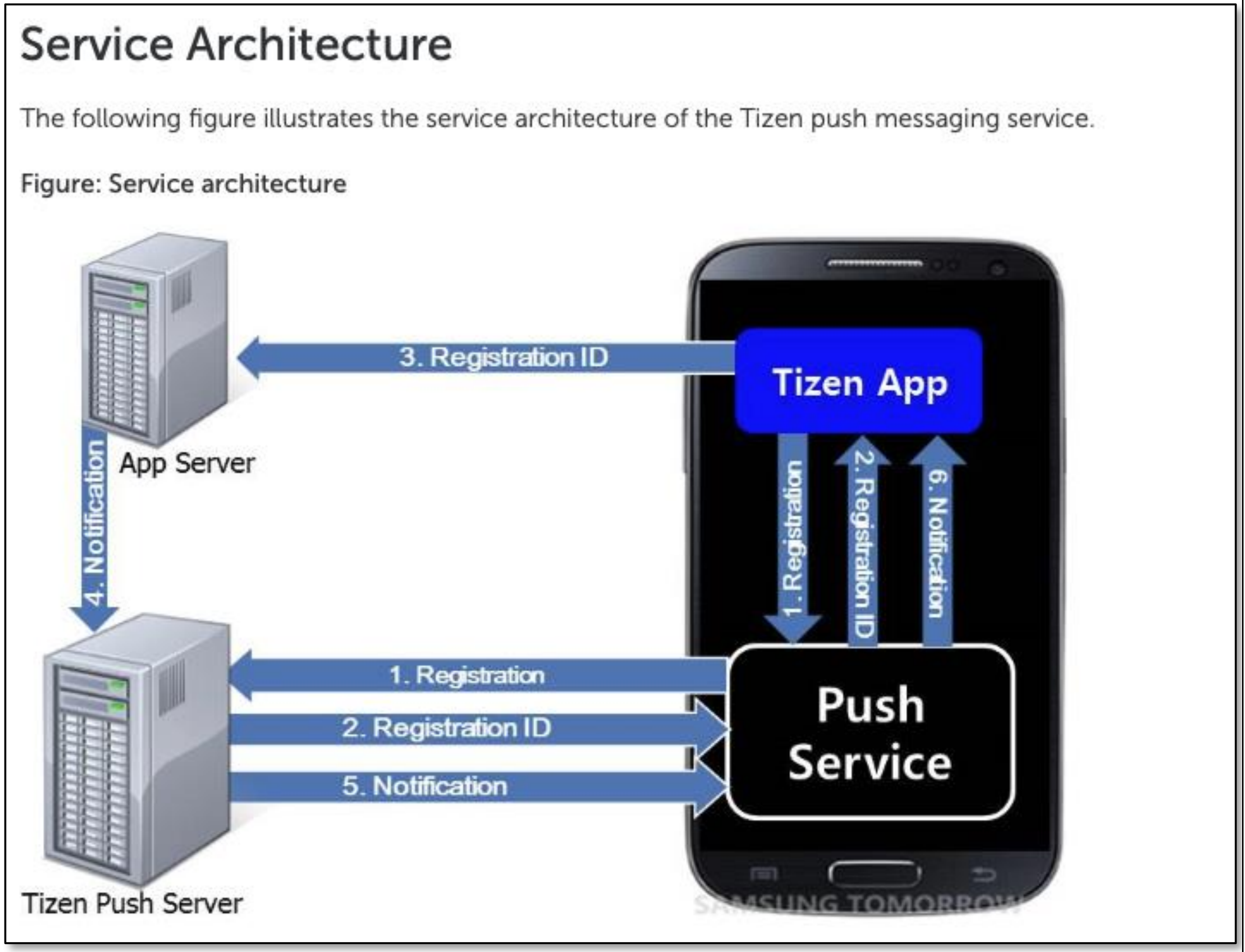
As another example, Samsung's Tizen based devices include memory that stores an encryption key that is shared between the agent and the network system. *See e.g.*,

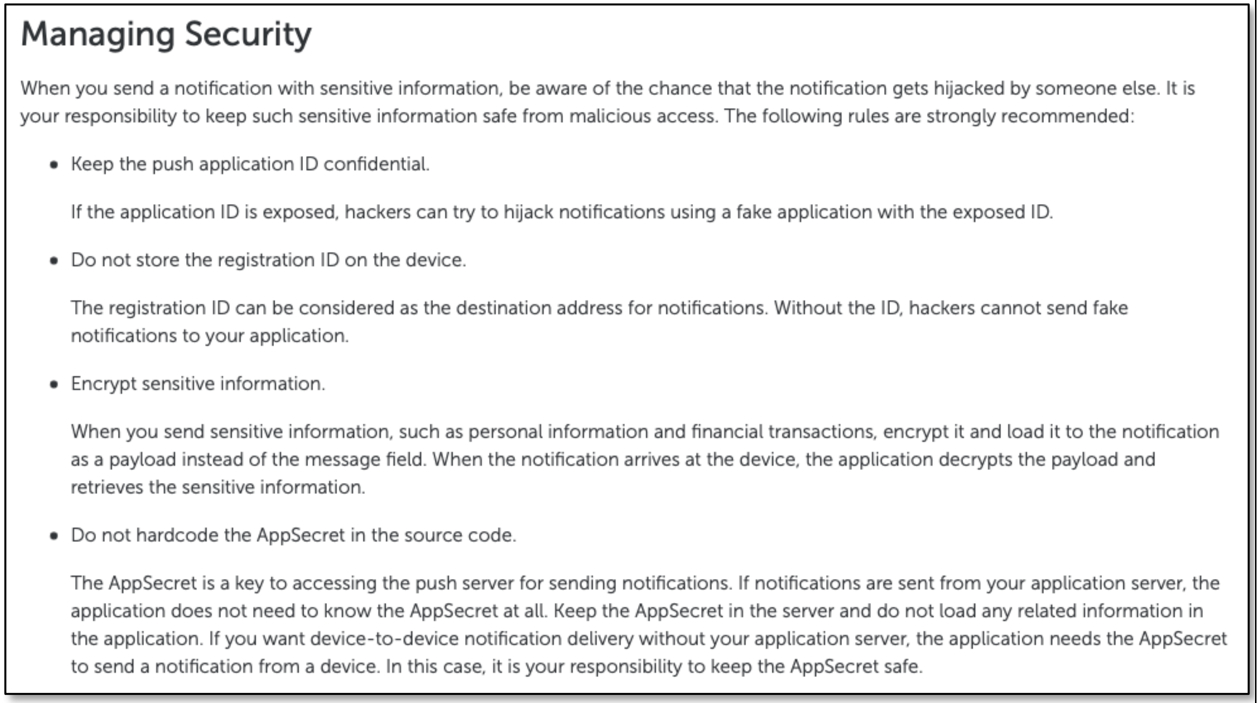


Wireless Connectivity

Wi-Fi ?	Wi-Fi Direct	Bluetooth ?
Yes (WiFi5)	Yes	Yes (BT5.2)

	https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/#specs;  **Wireless security protocols** The TV only supports the following wireless network security protocols. The TV cannot connect to non-certified wireless access point. – Authentication Modes: WEP, WPAPSK, WPA2PSK – Encryption Types: WEP, TKIP, AES https://downloadcenter.samsung.com/content/UM/202203/20220303092001587/OSNATSCB-3.2.0_EM_Oscar_Pontus_Nike_Kant-SU2e_USA_ENG_220216.0.pdf;

	 **Service Architecture** The following figure illustrates the service architecture of the Tizen push messaging service. Figure: Service architecture 3. Registration ID Tizen App App Server 4. Notification 1. Registration 2. Registration ID 6. Notification 1. Registration 2. Registration ID 5. Notification Push Service Tizen Push Server SAMSUNG TOMORROW



Managing Security

When you send a notification with sensitive information, be aware of the chance that the notification gets hijacked by someone else. It is your responsibility to keep such sensitive information safe from malicious access. The following rules are strongly recommended:

- Keep the push application ID confidential.

  If the application ID is exposed, hackers can try to hijack notifications using a fake application with the exposed ID.

- Do not store the registration ID on the device.

  The registration ID can be considered as the destination address for notifications. Without the ID, hackers cannot send fake notifications to your application.

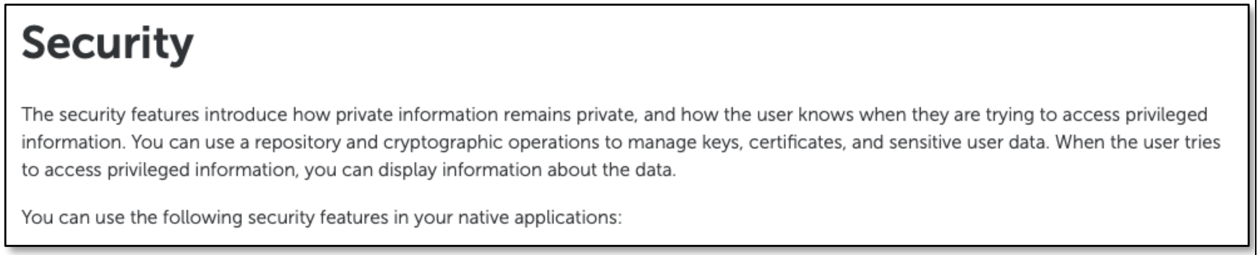
- Encrypt sensitive information.

  When you send sensitive information, such as personal information and financial transactions, encrypt it and load it to the notification as a payload instead of the message field. When the notification arrives at the device, the application decrypts the payload and retrieves the sensitive information.

- Do not hardcode the AppSecret in the source code.

  The AppSecret is a key to accessing the push server for sending notifications. If notifications are sent from your application server, the application does not need to know the AppSecret at all. Keep the AppSecret in the server and do not load any related information in the application. If you want device-to-device notification delivery without your application server, the application needs the AppSecret to send a notification from a device. In this case, it is your responsibility to keep the AppSecret safe.

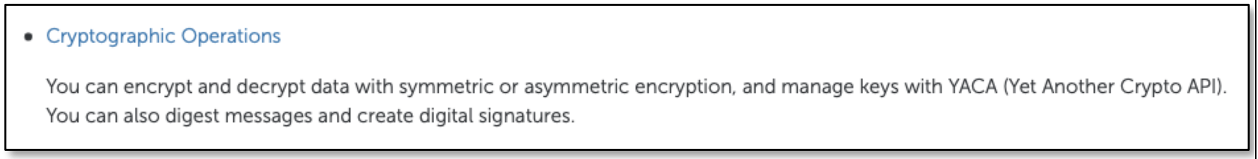
https://docs.tizen.org/application/native/guides/messaging/push/#connect;



Security

The security features introduce how private information remains private, and how the user knows when they are trying to access privileged information. You can use a repository and cryptographic operations to manage keys, certificates, and sensitive user data. When the user tries to access privileged information, you can display information about the data.

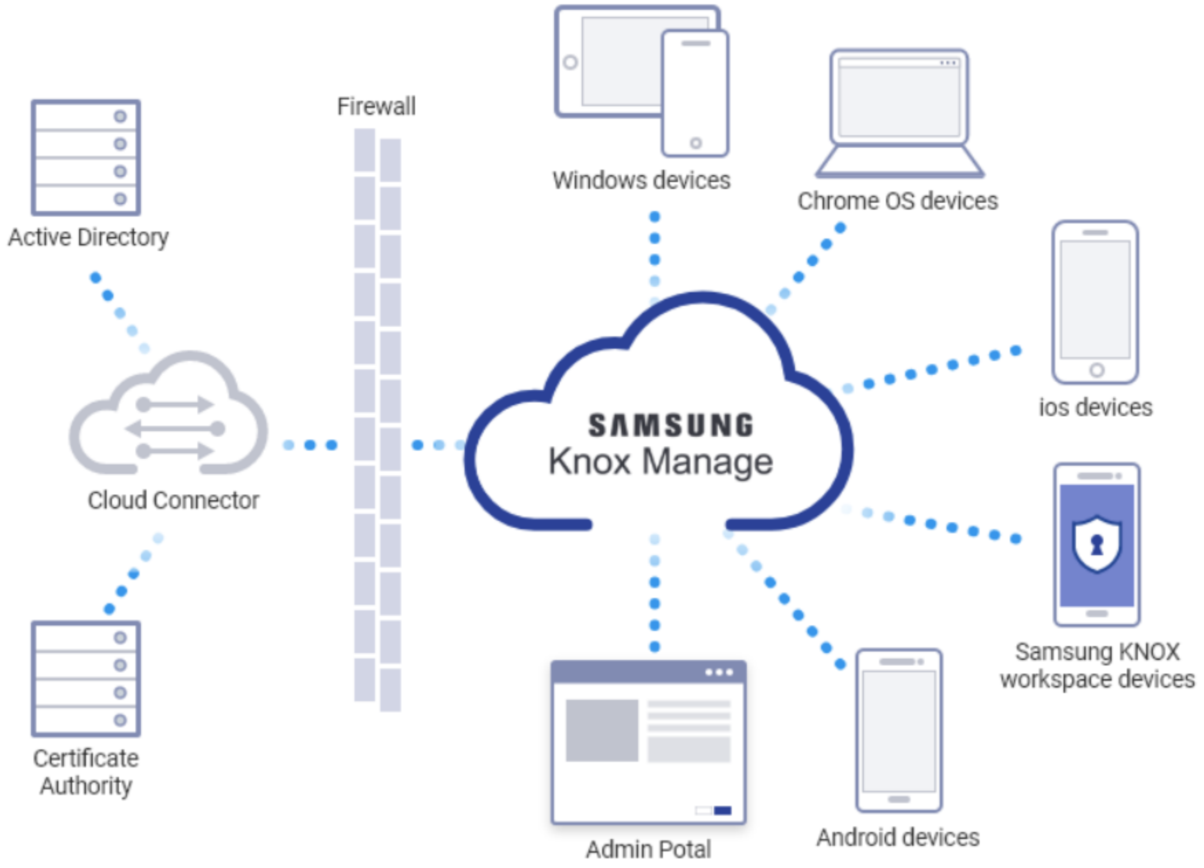
You can use the following security features in your native applications:

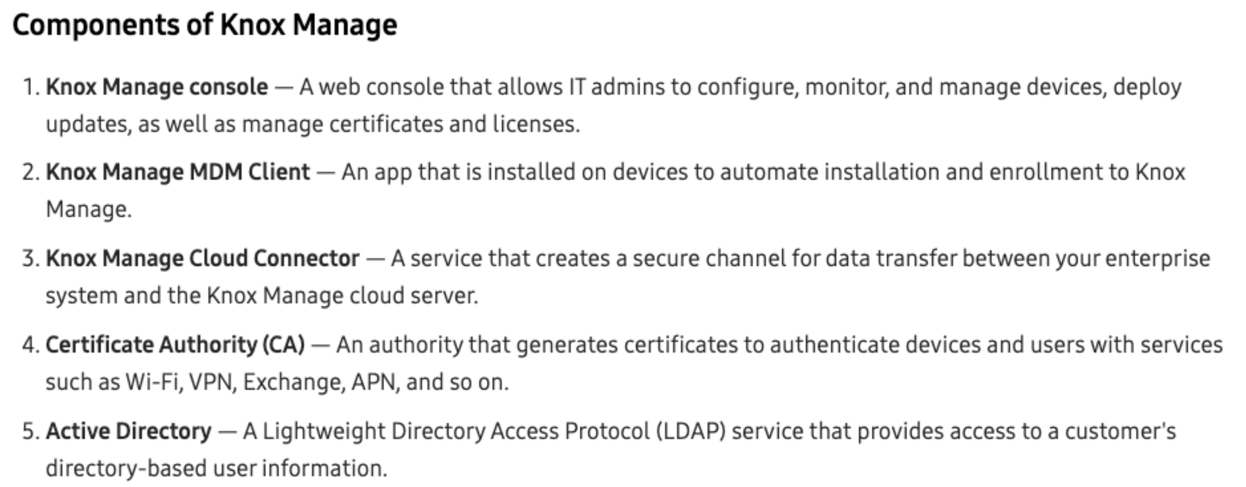


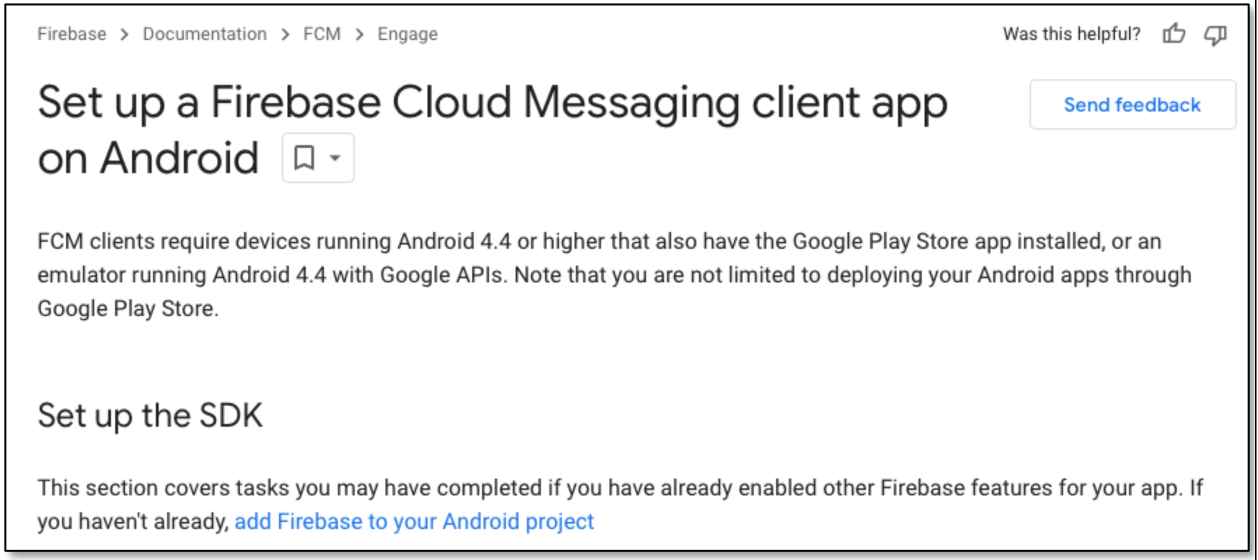
- Cryptographic Operations

  You can encrypt and decrypt data with symmetric or asymmetric encryption, and manage keys with YACA (Yet Another Crypto API). You can also digest messages and create digital signatures.

https://docs.tizen.org/application/native/guides/security/overview/.

	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, include memory that stores an encryption key that is shared between the agent and the network system.  Firewall Active Directory Cloud Connector Certificate Authority Windows devices Chrome OS devices ios devices SAMSUNG Knox Manage Samsung KNOX workspace devices Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	**Components of Knox Manage** 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
1[d] wherein the service control device link agent is configured to: receive, over the service control link, an encrypted agent message from the service control server link element,	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise a "service control device link agent [which] is configured to: receive, over the service control link, an encrypted agent message from the service control server link element." For example, Samsung Galaxy phones and tablets receive encrypted messages from a server. *See e.g.*,

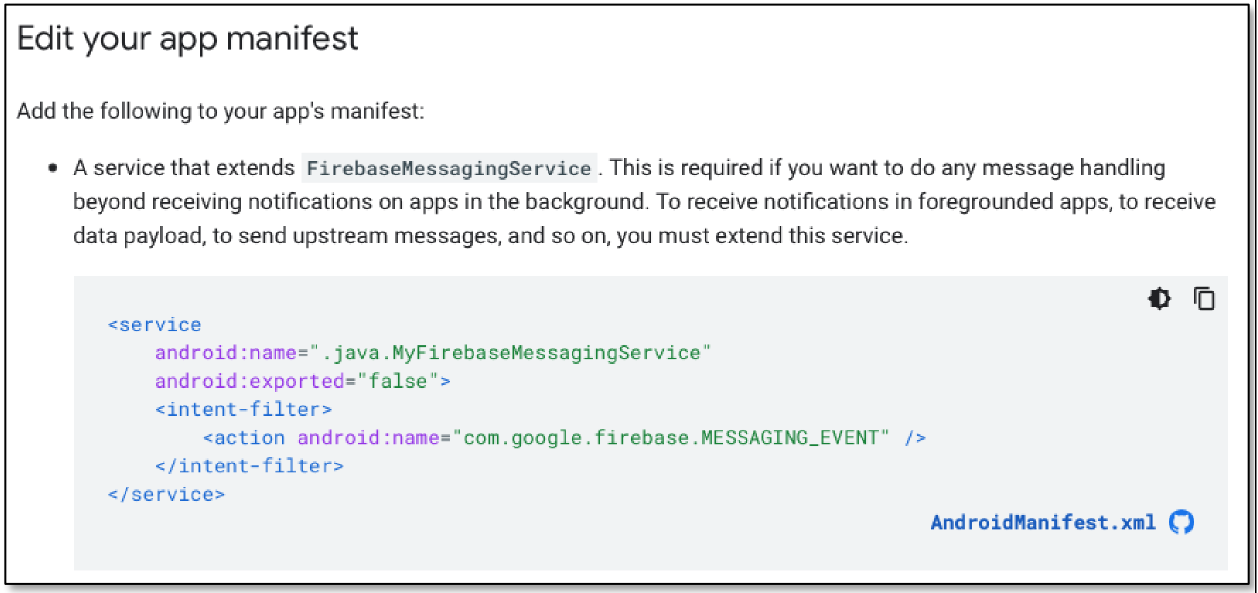


Set up a Firebase Cloud Messaging client app on Android

FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store.

Set up the SDK

This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project

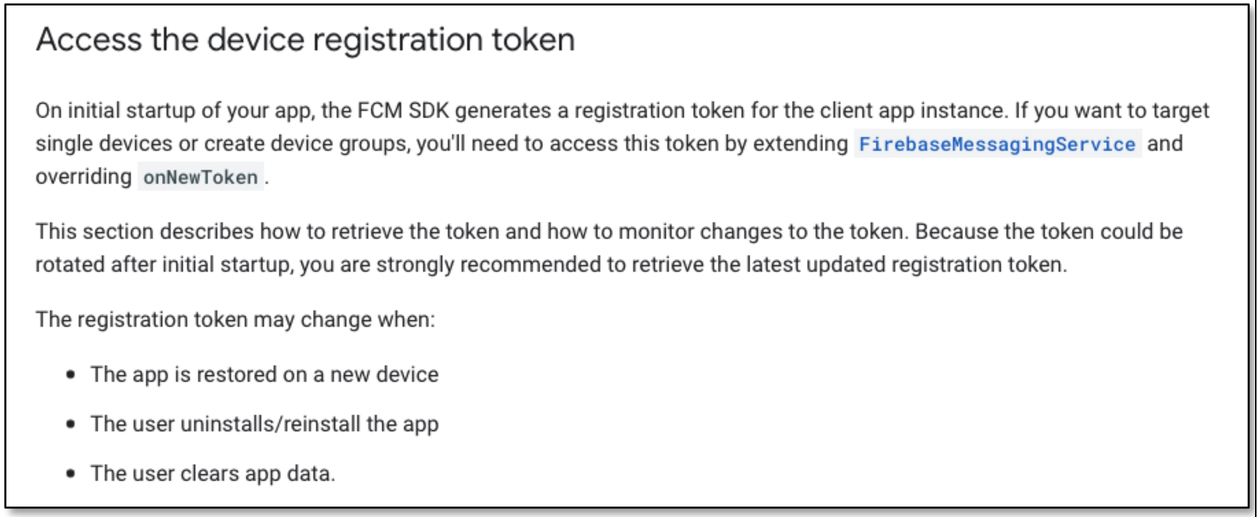


Edit your app manifest

Add the following to your app's manifest:

- A service that extends `FirebaseMessagingService`. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service.

```xml
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```



Access the device registration token

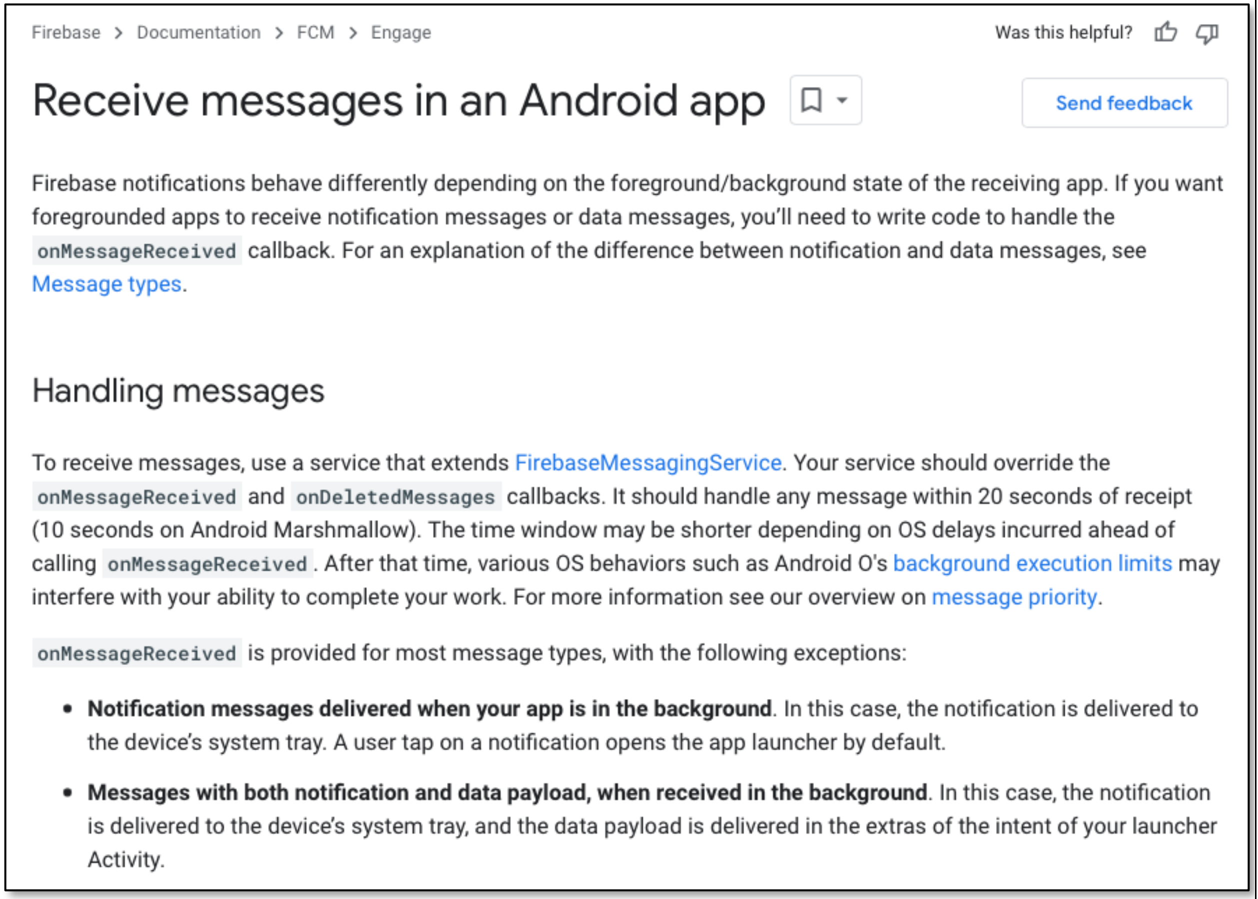
On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`.

This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

https://firebase.google.com/docs/cloud-messaging/android/client;



Receive messages in an Android app

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see Message types.

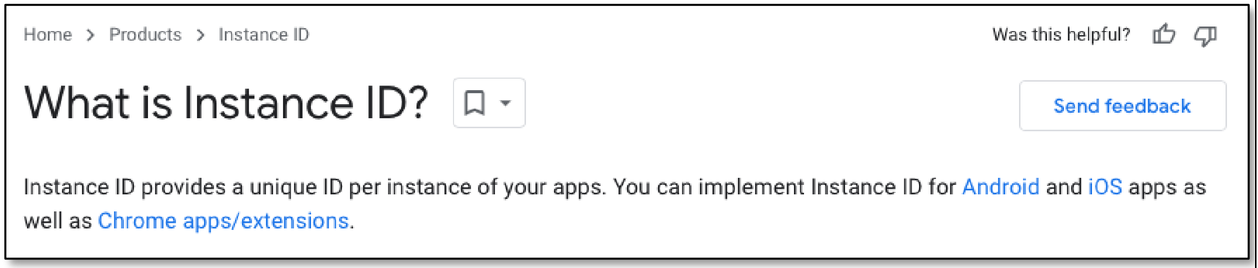
Handling messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's background execution limits may interfere with your ability to complete your work. For more information see our overview on message priority.

`onMessageReceived` is provided for most message types, with the following exceptions:

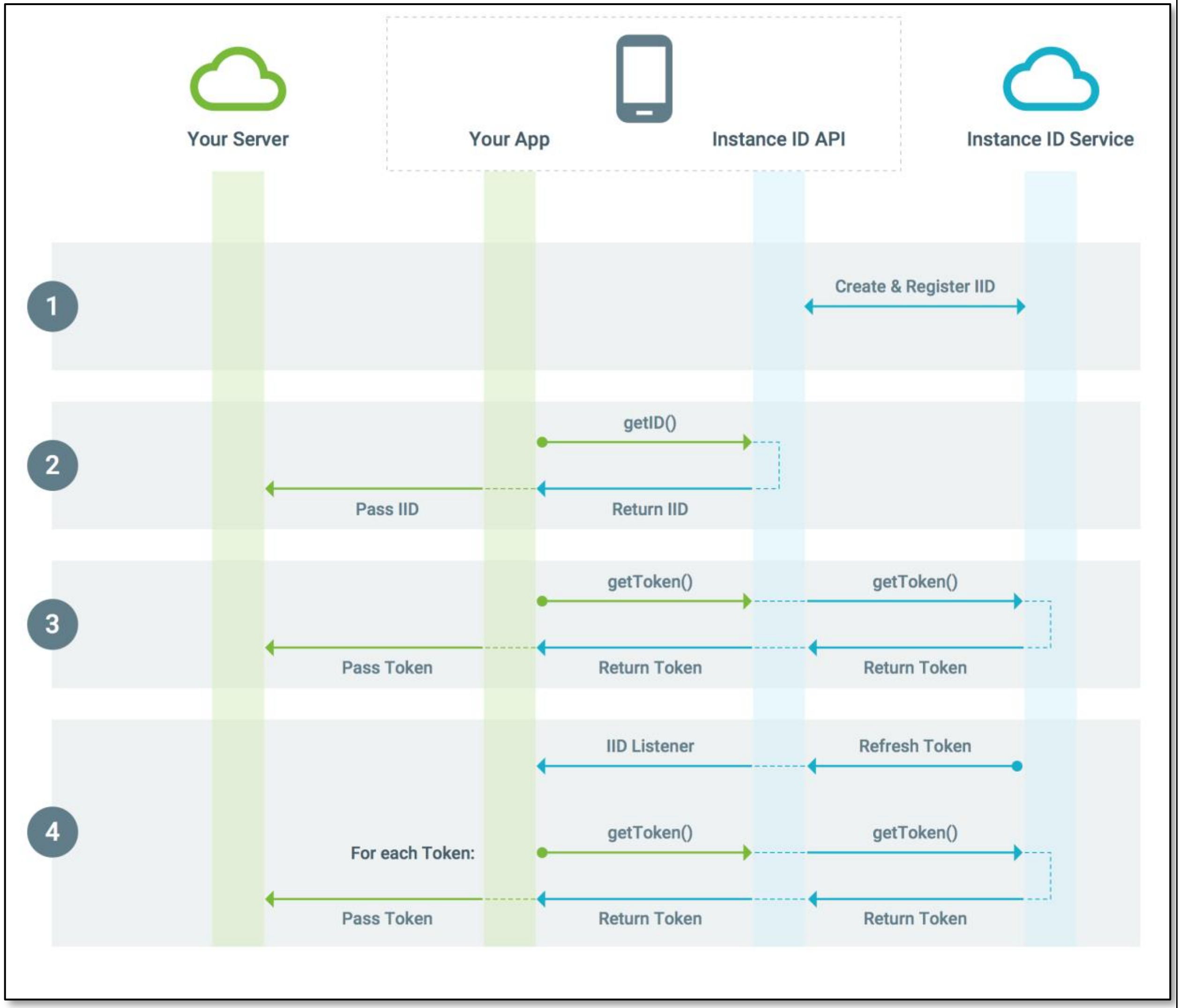
- **Notification messages delivered when your app is in the background**. In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

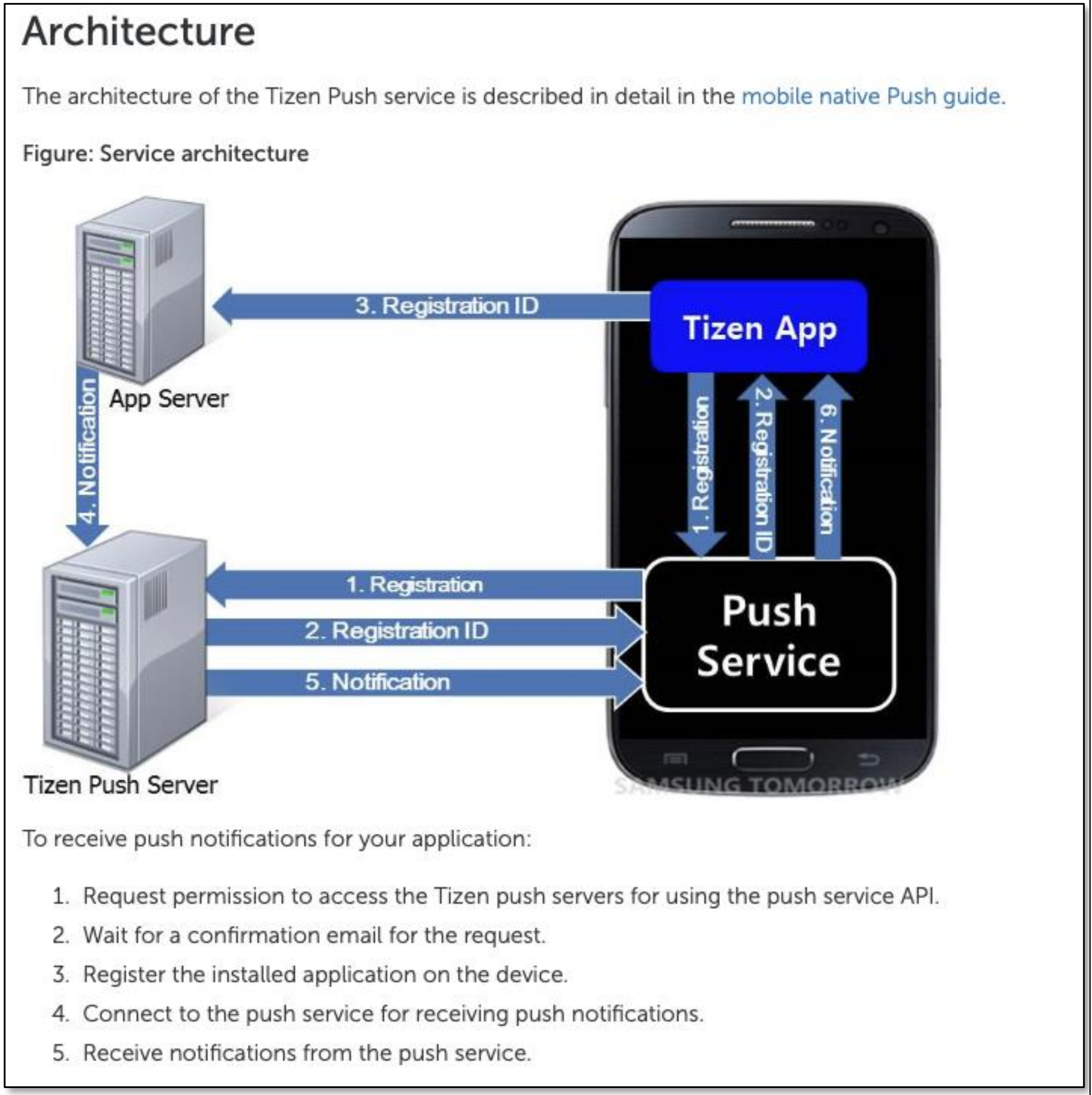
https://firebase.google.com/docs/cloud-messaging/android/receive;



What is Instance ID?

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for Android and iOS apps as well as Chrome apps/extensions.

	 Your Server Your App Instance ID API Instance ID Service 1 Create & Register IID 2 getID() Pass IID Return IID 3 getToken() getToken() Pass Token Return Token Return Token 4 IID Listener Refresh Token For each Token: getToken() getToken() Pass Token Return Token Return Token https://developers.google.com/instance-id. For further example, Samsung's Tizen based devices receive encrypted messages from a server. *See e.g.*,



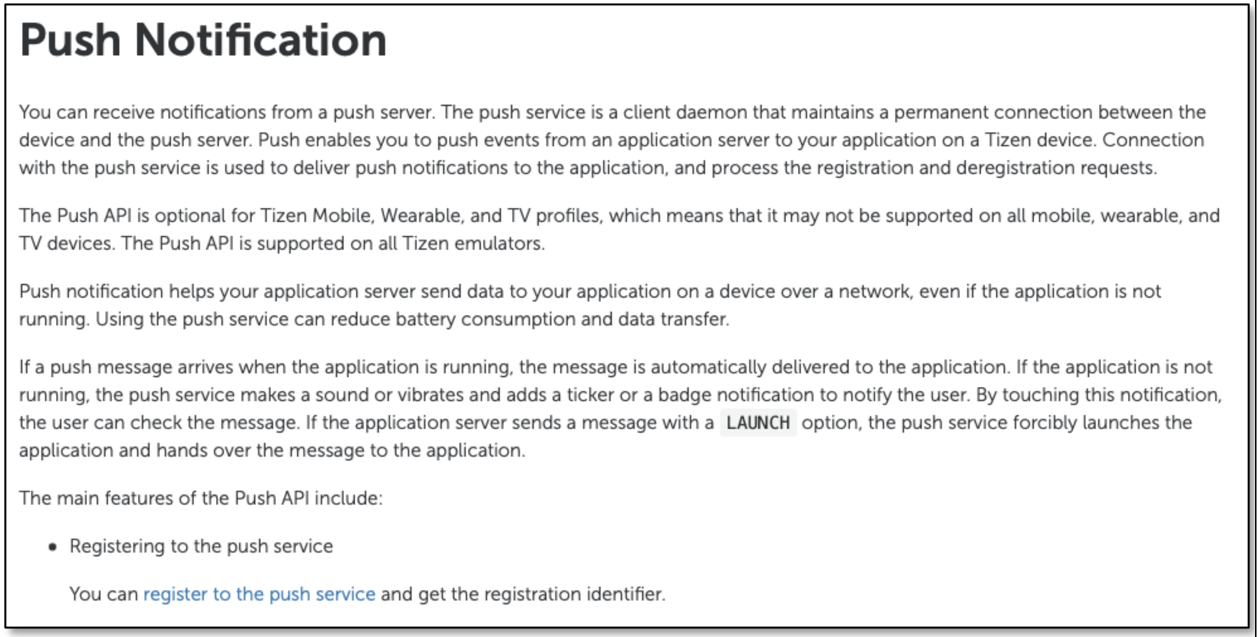
Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture

To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.



Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

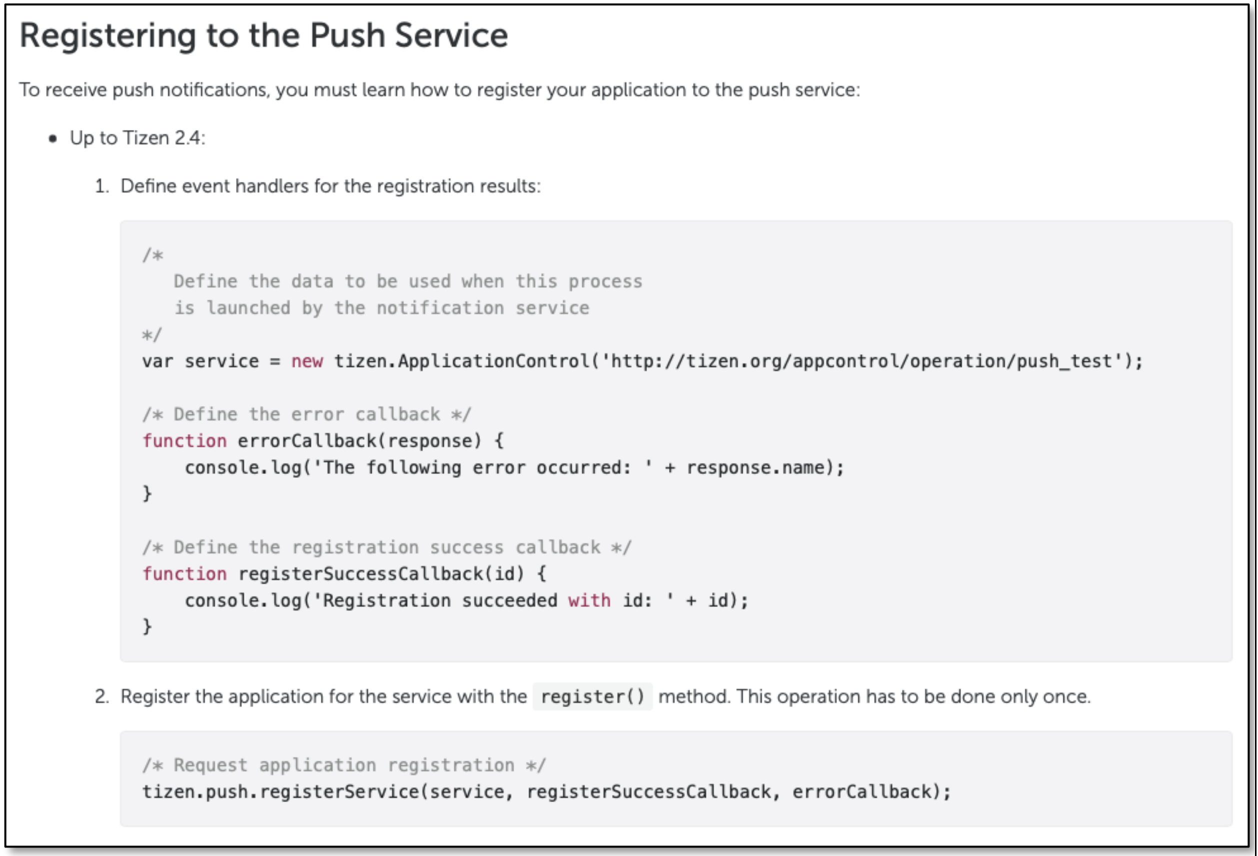
Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

  You can register to the push service and get the registration identifier.



Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

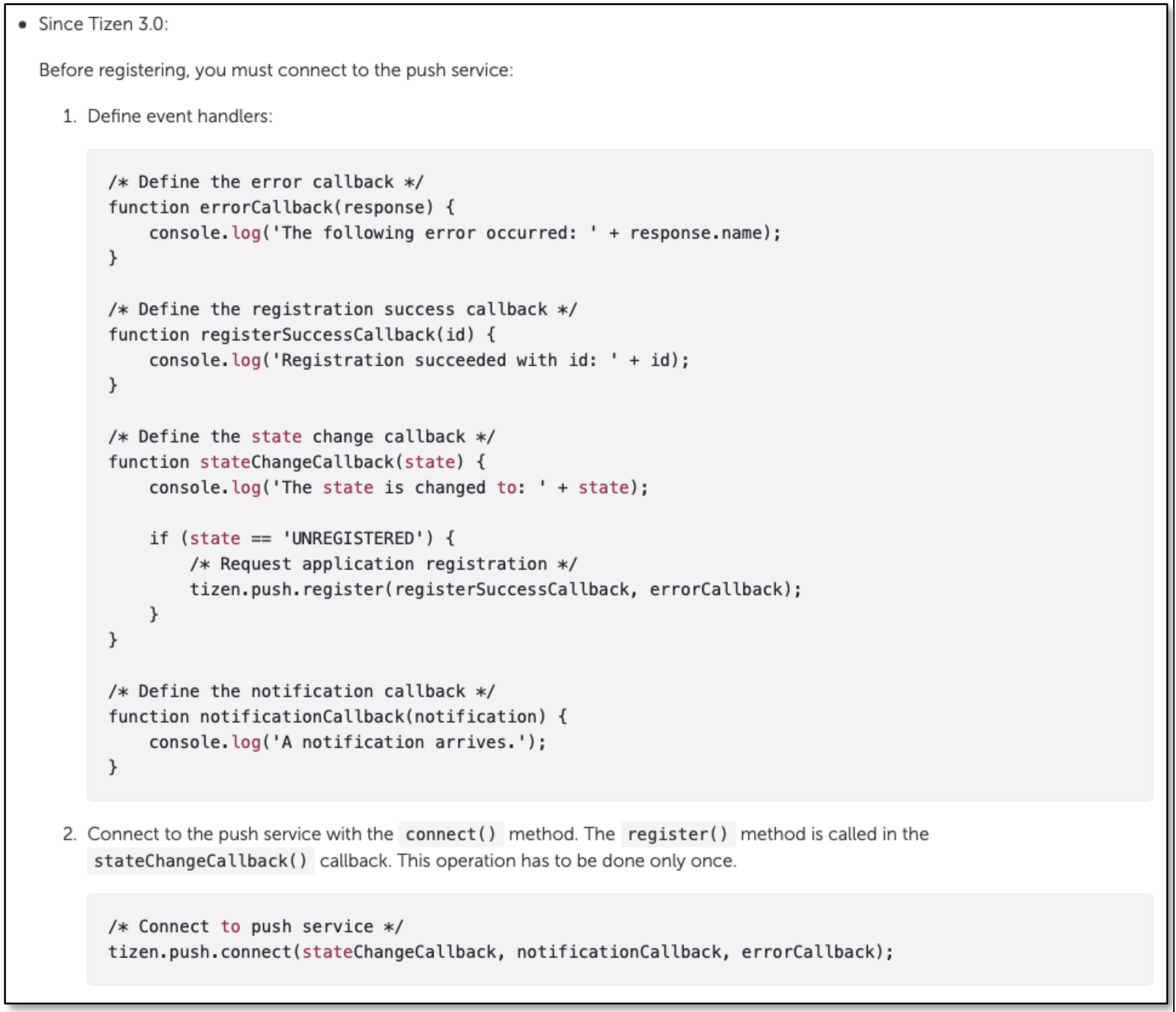
- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

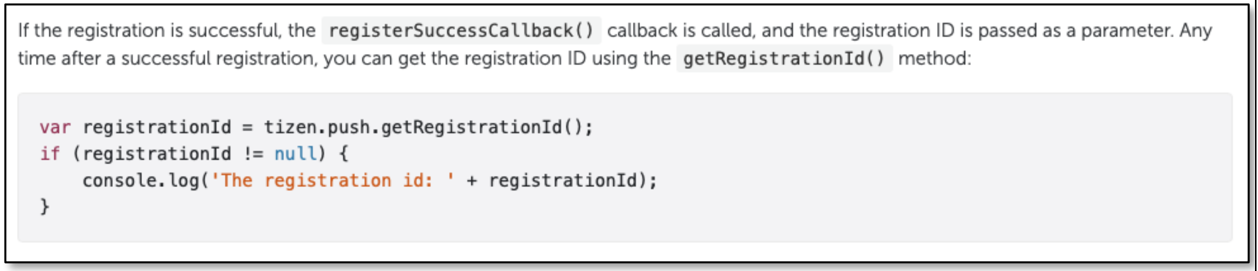
  Before registering, you must connect to the push service:

  1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

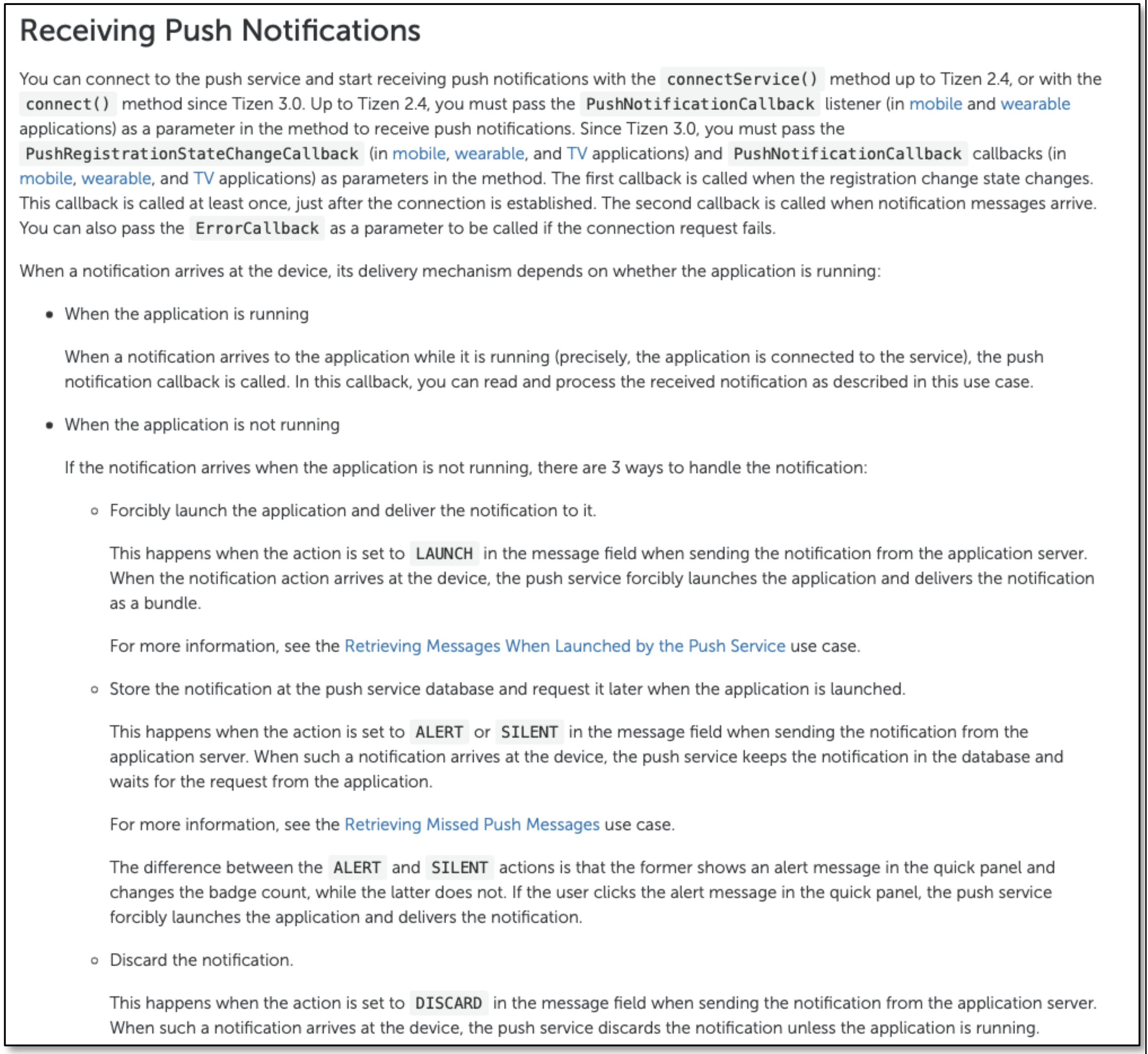
  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```



If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```



Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in mobile and wearable applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the `PushRegistrationStateChangeCallback` (in mobile, wearable, and TV applications) and `PushNotificationCallback` callbacks (in mobile, wearable, and TV applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

  When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

  If the notification arrives when the application is not running, there are 3 ways to handle the notification:

  - Forcibly launch the application and deliver the notification to it.

    This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

  - Store the notification at the push service database and request it later when the application is launched.

    This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

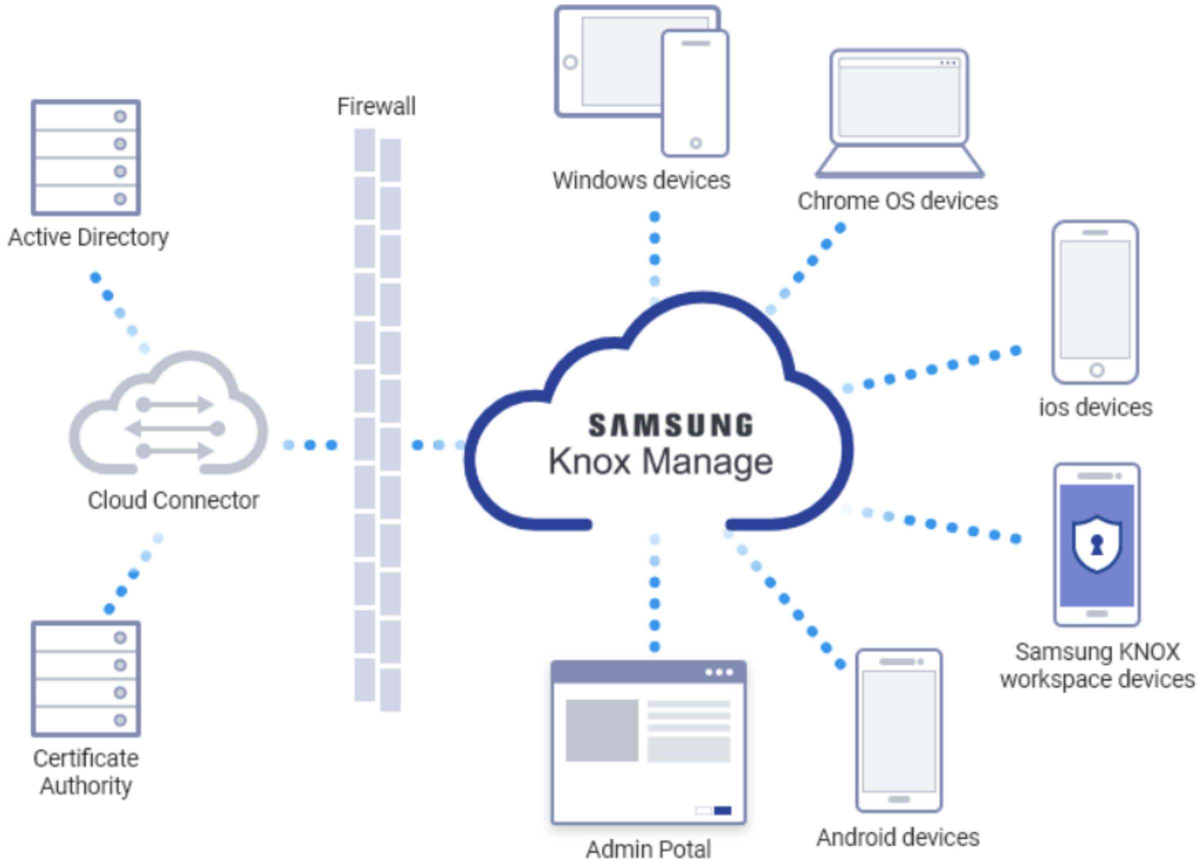
    For more information, see the Retrieving Missed Push Messages use case.

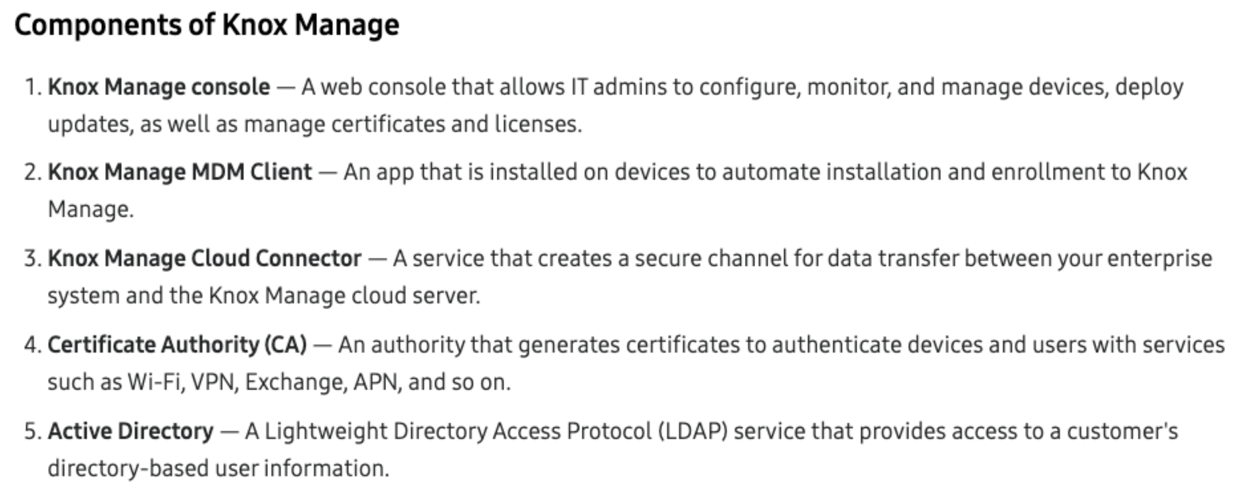
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

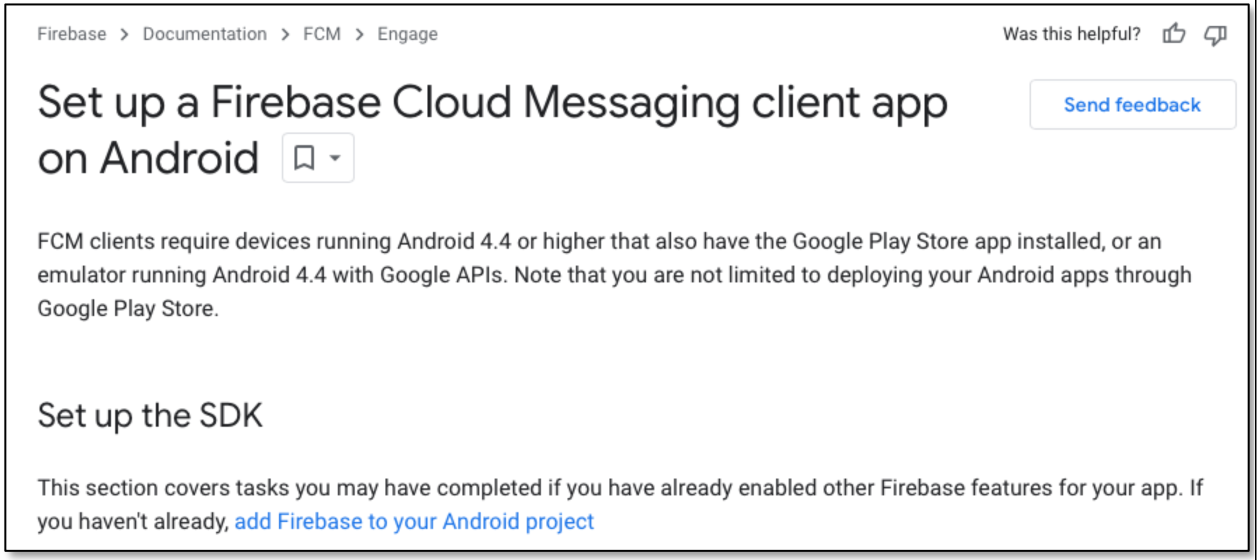
  - Discard the notification.

    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

https://docs.tizen.org/application/web/guides/messaging/push/.

	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, receive encrypted messages from a server.  Firewall Active Directory Windows devices Chrome OS devices ios devices SAMSUNG Knox Manage Cloud Connector Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 **Components of Knox Manage** 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
1[e] using the encryption key, obtain a decrypted agent message, the decrypted agent message comprising a particular agent identifier and message content for delivery to a particular device agent of the plurality of device agents, the particular agent identifier identifying the particular device agent, the message content from a particular server of a plurality of servers communicatively coupled to the service control server link element, and	Samsung Galaxy phones and tablets and Samsung's Tizen based devices "us[e] the encryption key, obtain a decrypted agent message, the decrypted agent message comprising a particular agent identifier and message content for delivery to a particular device agent of the plurality of device agents, the particular agent identifier identifying the particular device agent, the message content from a particular server of a plurality of servers communicatively coupled to the service control server link element." For example, Samsung Galaxy phones and tablets decrypt messages from a plurality of servers which have identifiers and content for delivery to particular device agents. *See e.g.*,

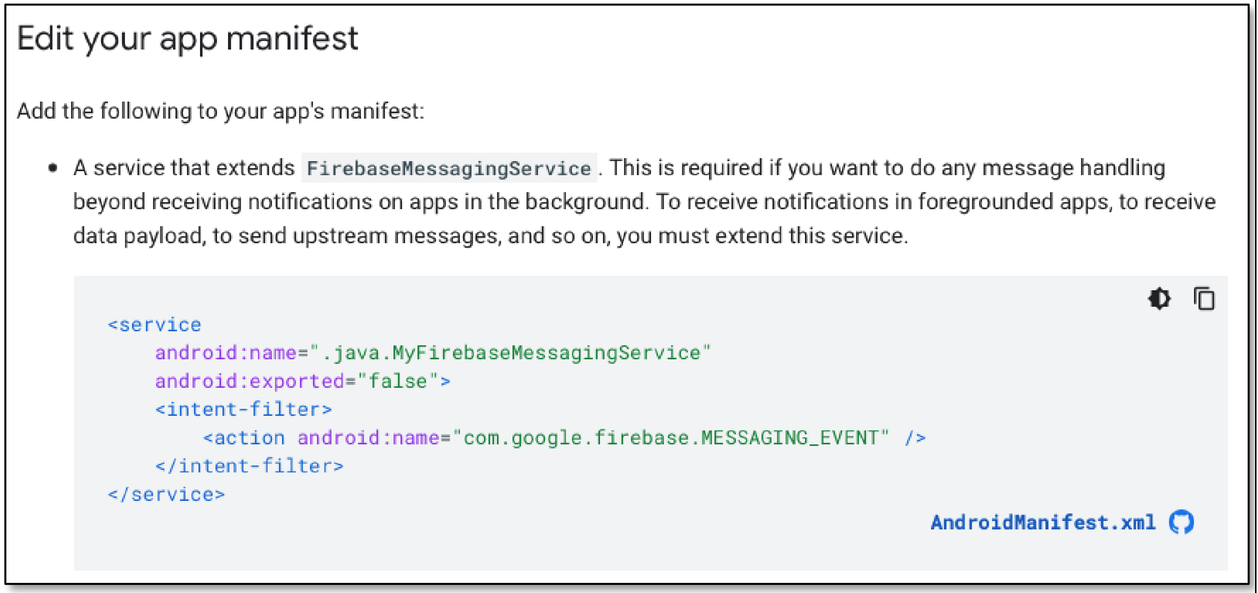


Set up a Firebase Cloud Messaging client app on Android

FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store.

Set up the SDK

This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project

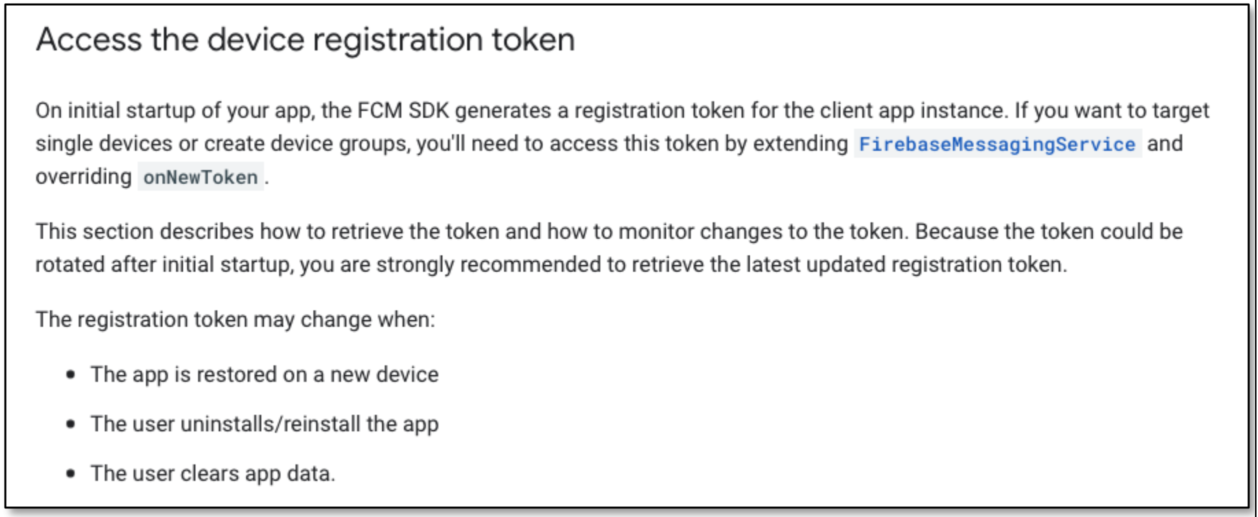


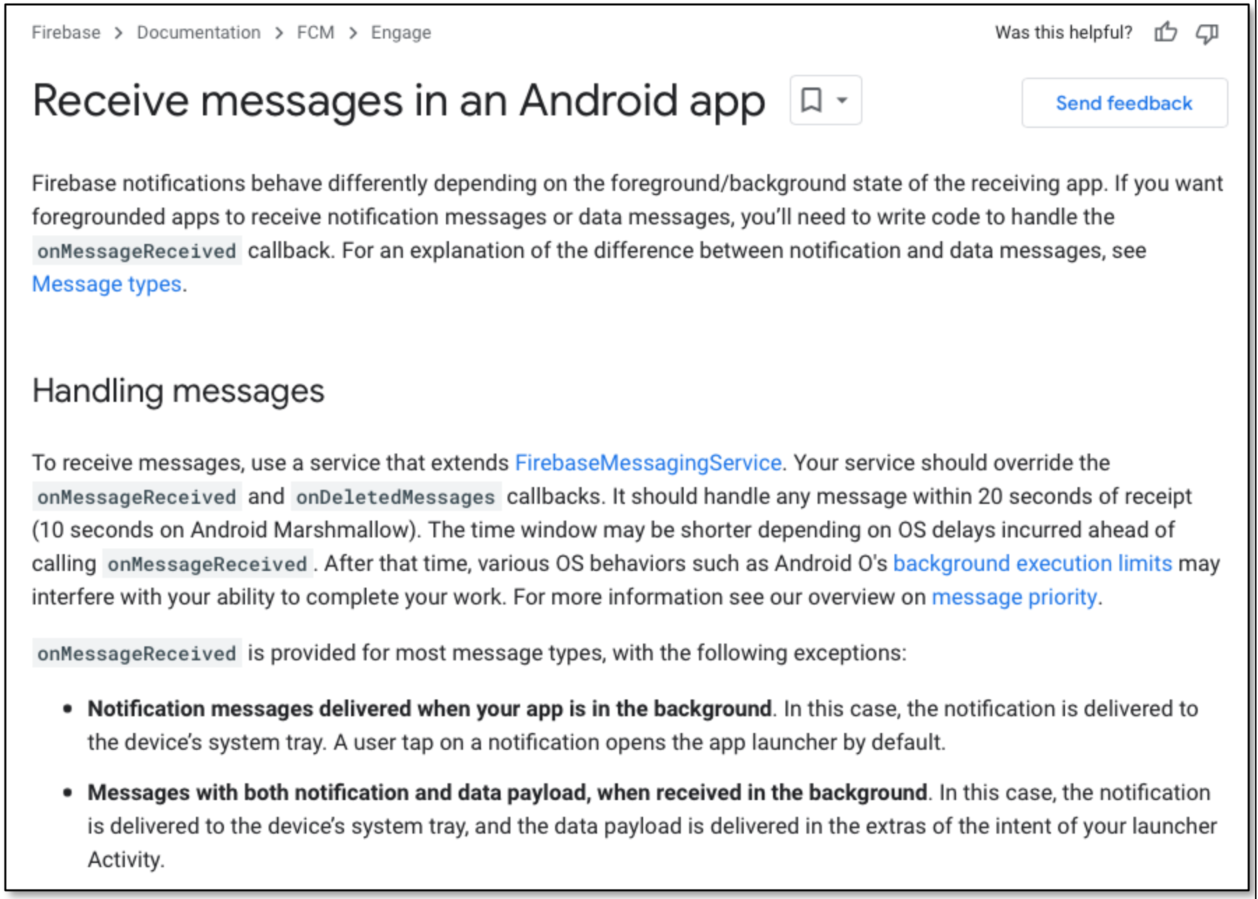
Edit your app manifest

Add the following to your app's manifest:

- A service that extends `FirebaseMessagingService`. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service.

```
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

	 **Access the device registration token** On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`. This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token. The registration token may change when: • The app is restored on a new device • The user uninstalls/reinstall the app • The user clears app data. https://firebase.google.com/docs/cloud-messaging/android/client;



Receive messages in an Android app

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see Message types.

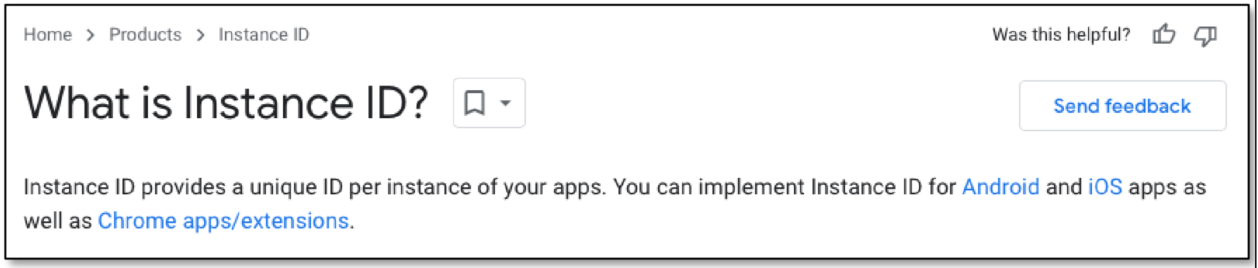
Handling messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's background execution limits may interfere with your ability to complete your work. For more information see our overview on message priority.

`onMessageReceived` is provided for most message types, with the following exceptions:

- **Notification messages delivered when your app is in the background**. In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

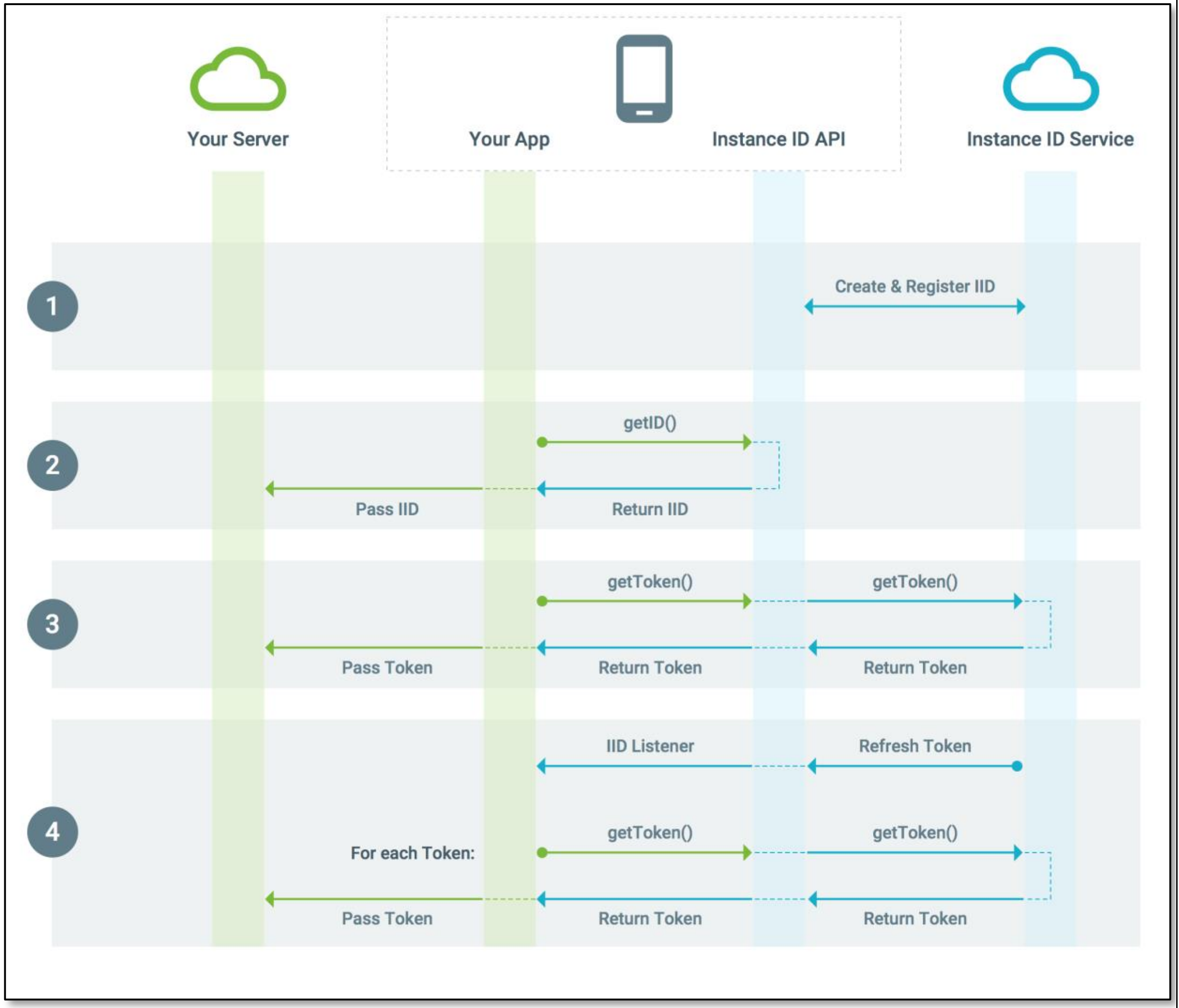
https://firebase.google.com/docs/cloud-messaging/android/receive;



What is Instance ID?

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for Android and iOS apps as well as Chrome apps/extensions.

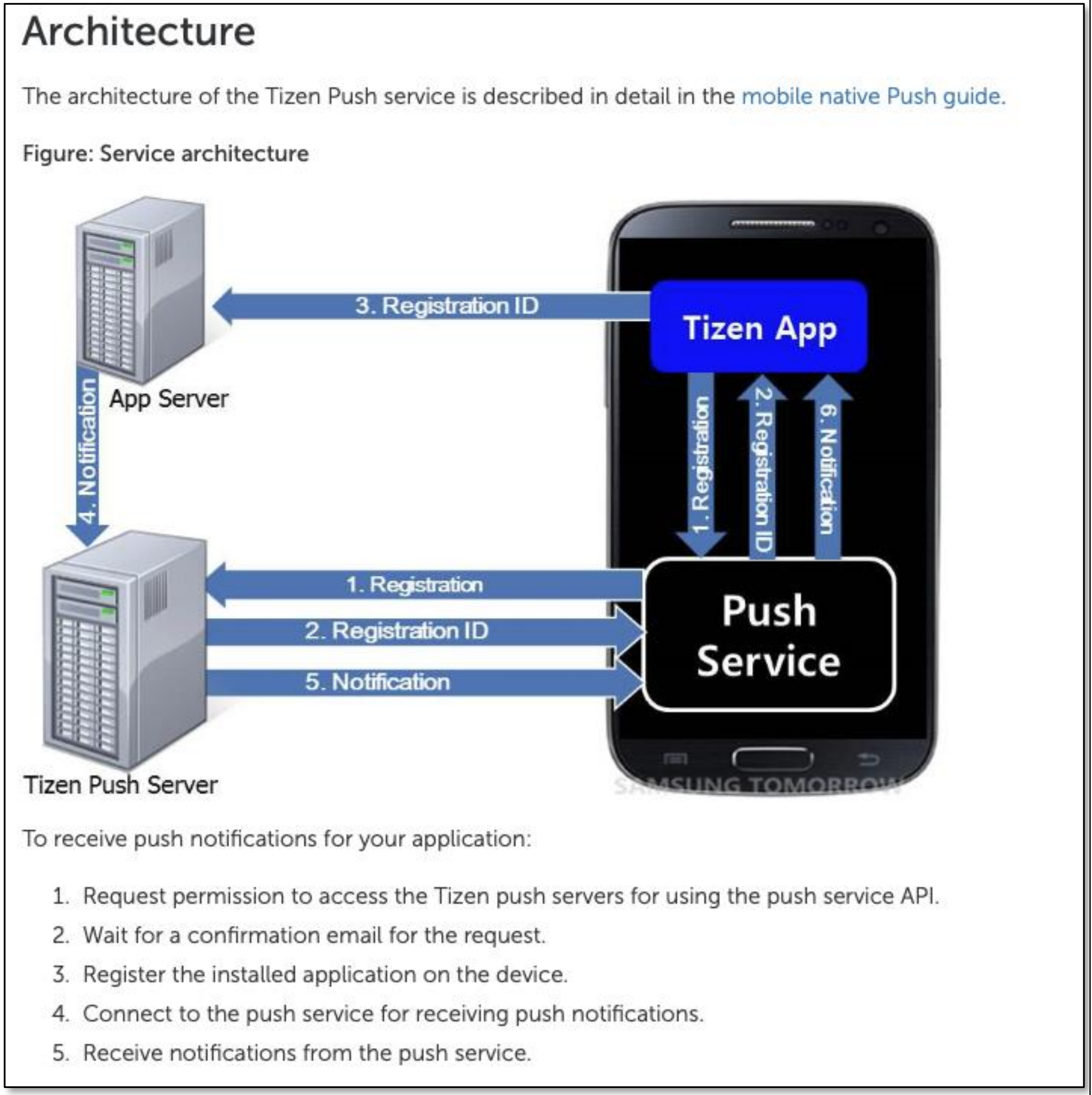
<table>
<tr>
<td></td>
<td>



https://developers.google.com/instance-id.

For further example, Samsung’s Tizen based devices receive encrypted messages from a server. *See e.g.*,

</td>
</tr>
</table>



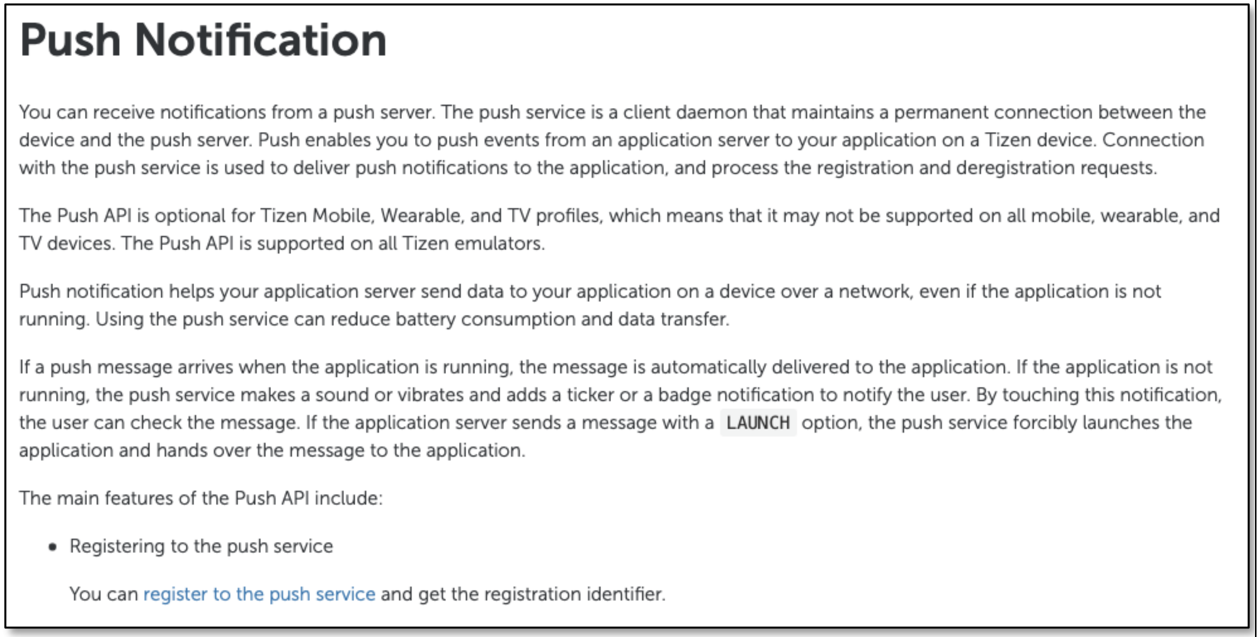
Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture

To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.



Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

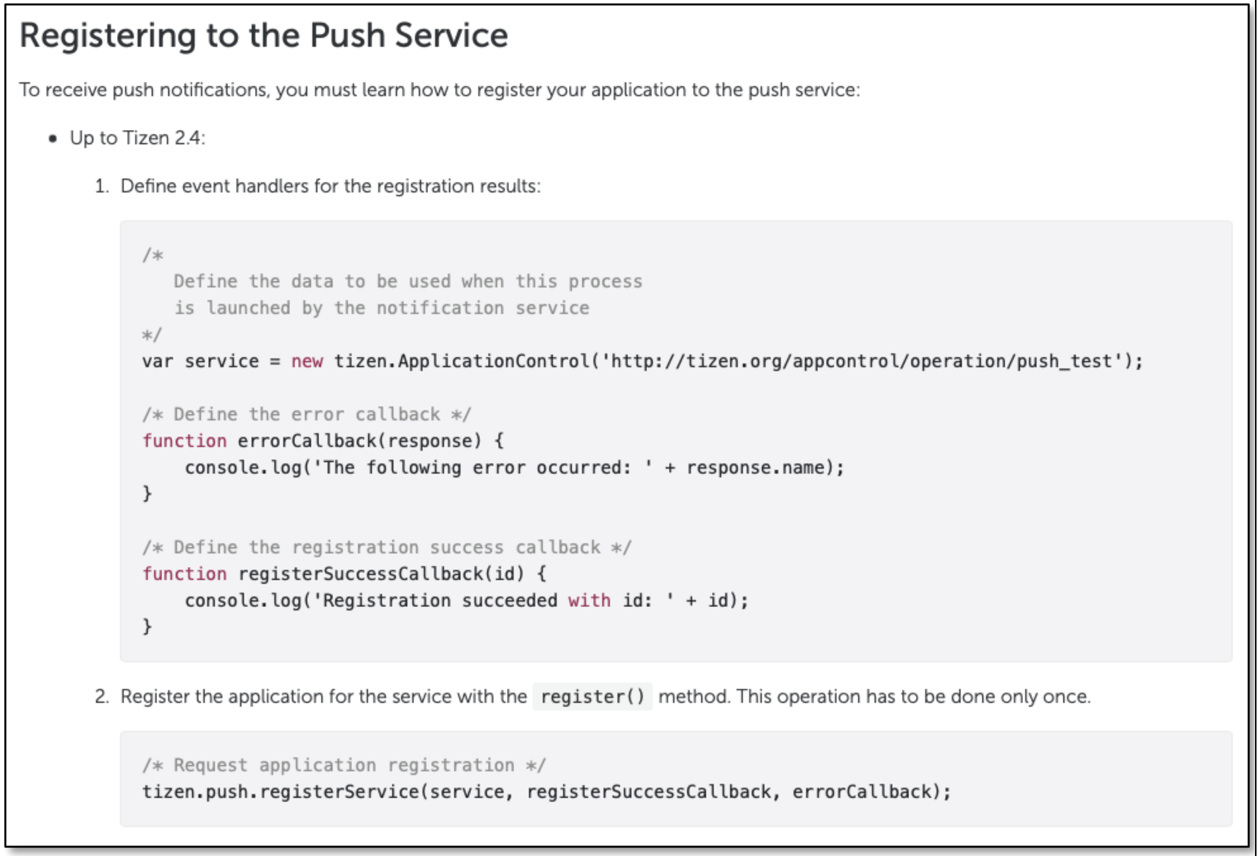
Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

  You can register to the push service and get the registration identifier.



Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

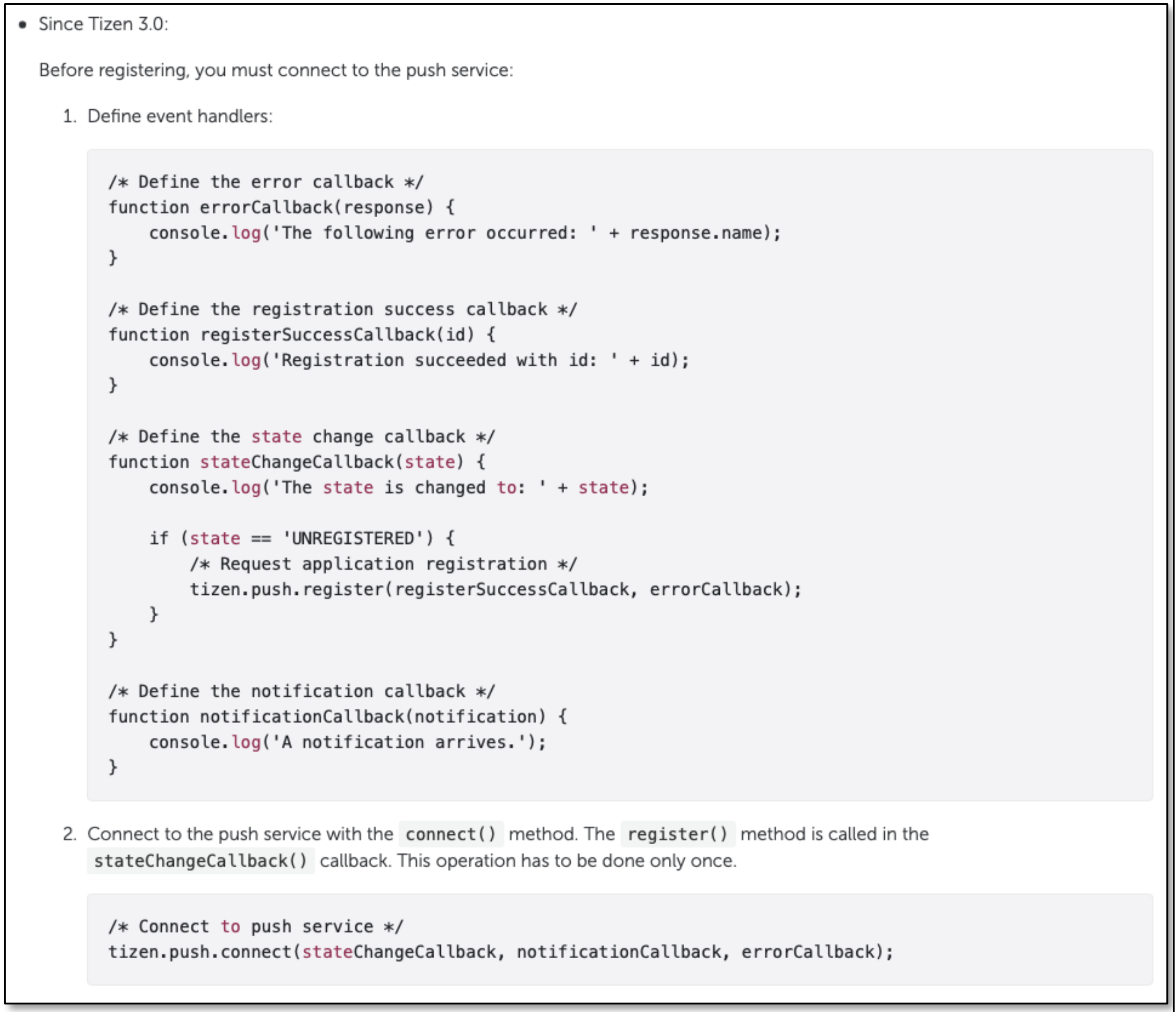
- Up to Tizen 2.4:

  1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

  2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

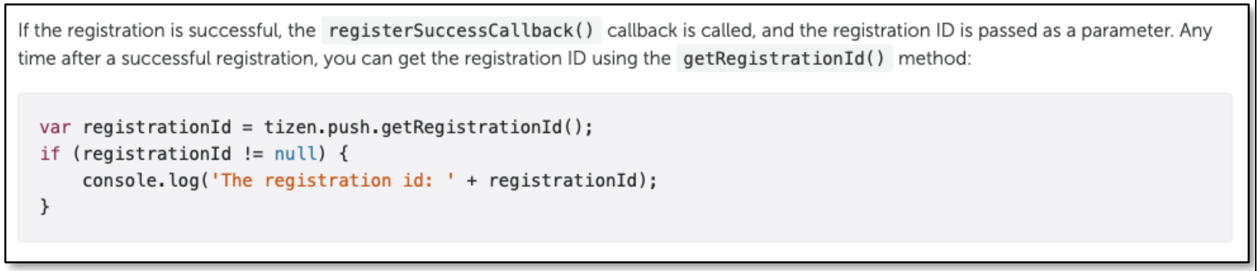
  Before registering, you must connect to the push service:

  1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

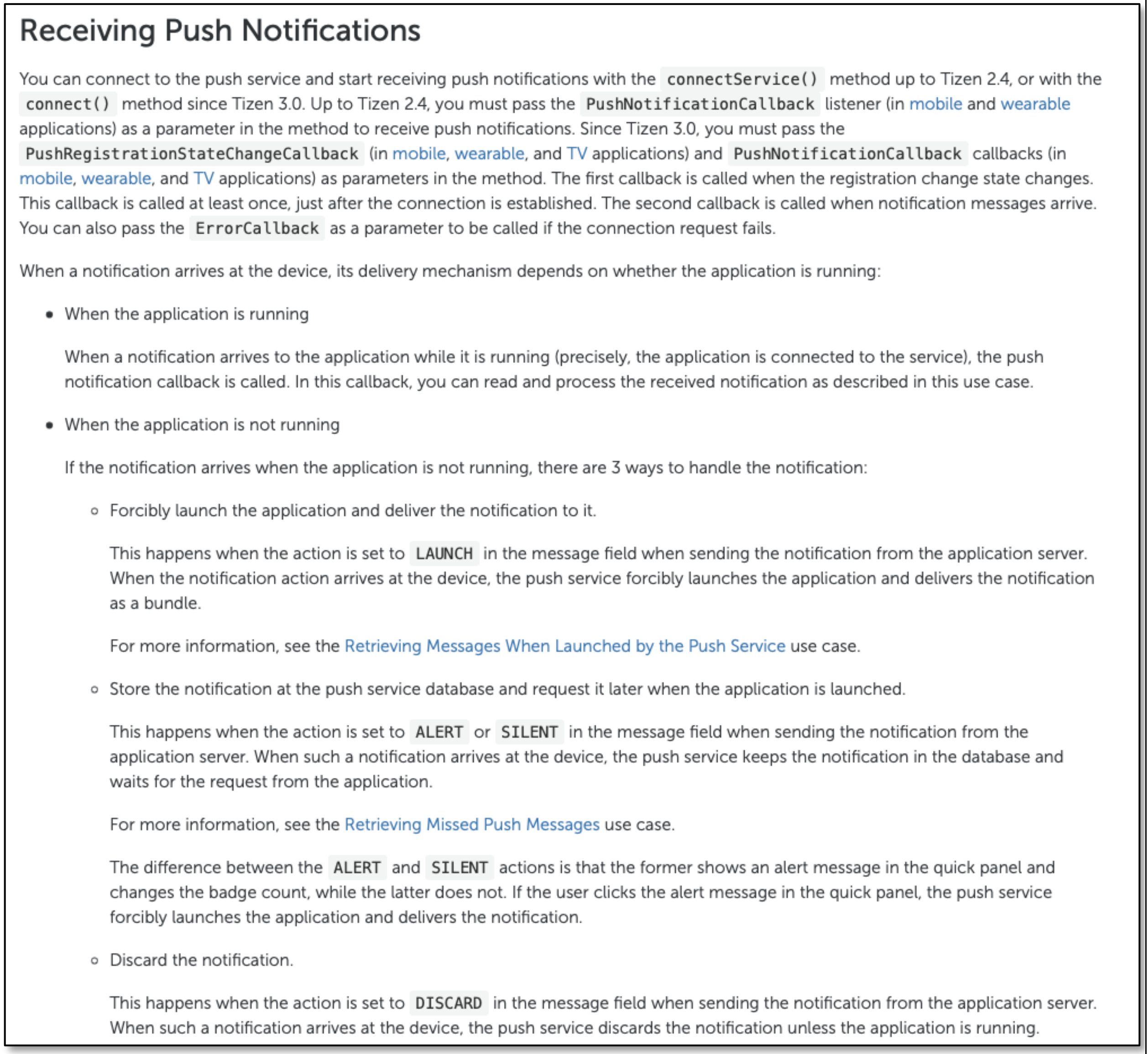
  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```



If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```



Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in mobile and wearable applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the `PushRegistrationStateChangeCallback` (in mobile, wearable, and TV applications) and `PushNotificationCallback` callbacks (in mobile, wearable, and TV applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

  When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

  If the notification arrives when the application is not running, there are 3 ways to handle the notification:

  - Forcibly launch the application and deliver the notification to it.

    This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

  - Store the notification at the push service database and request it later when the application is launched.

    This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

    For more information, see the Retrieving Missed Push Messages use case.

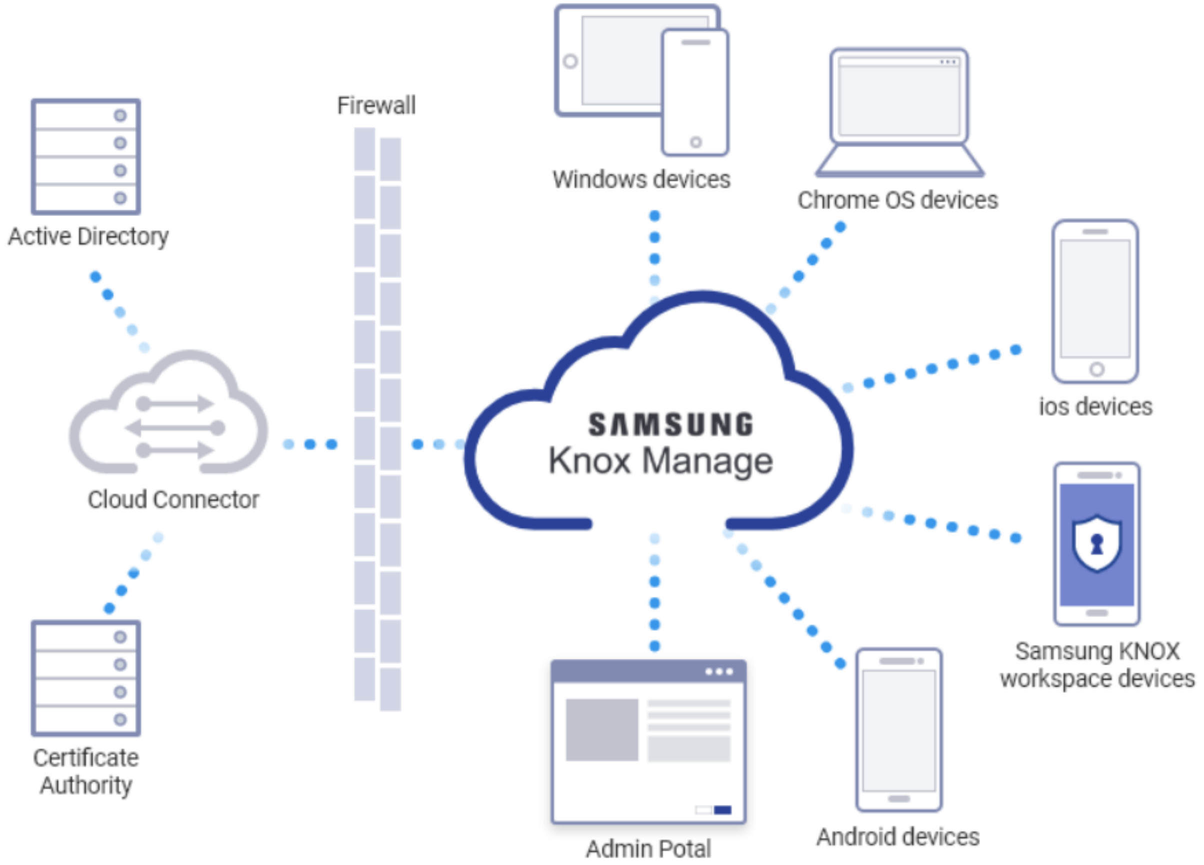
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

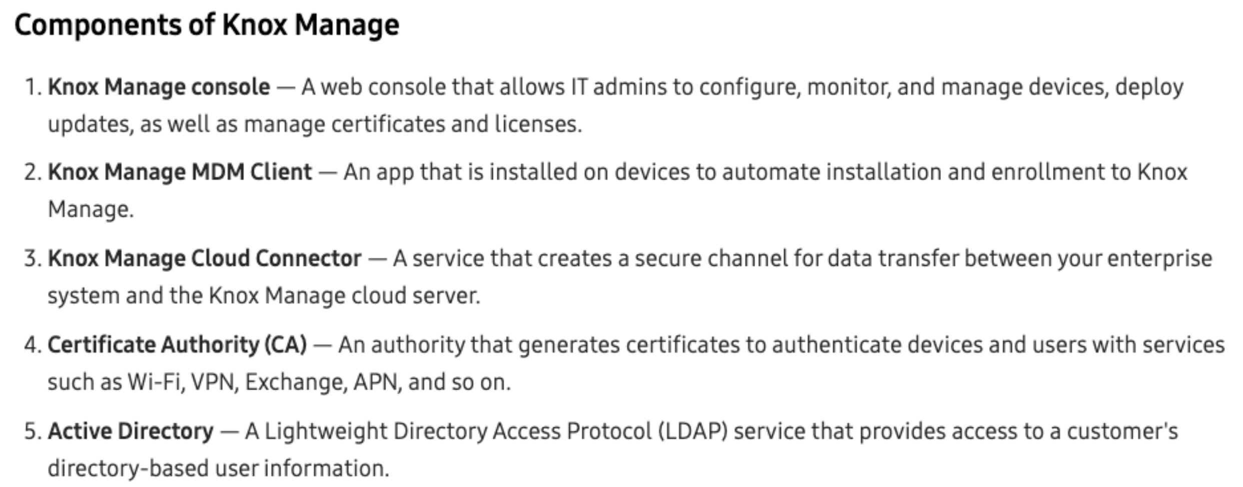
  - Discard the notification.

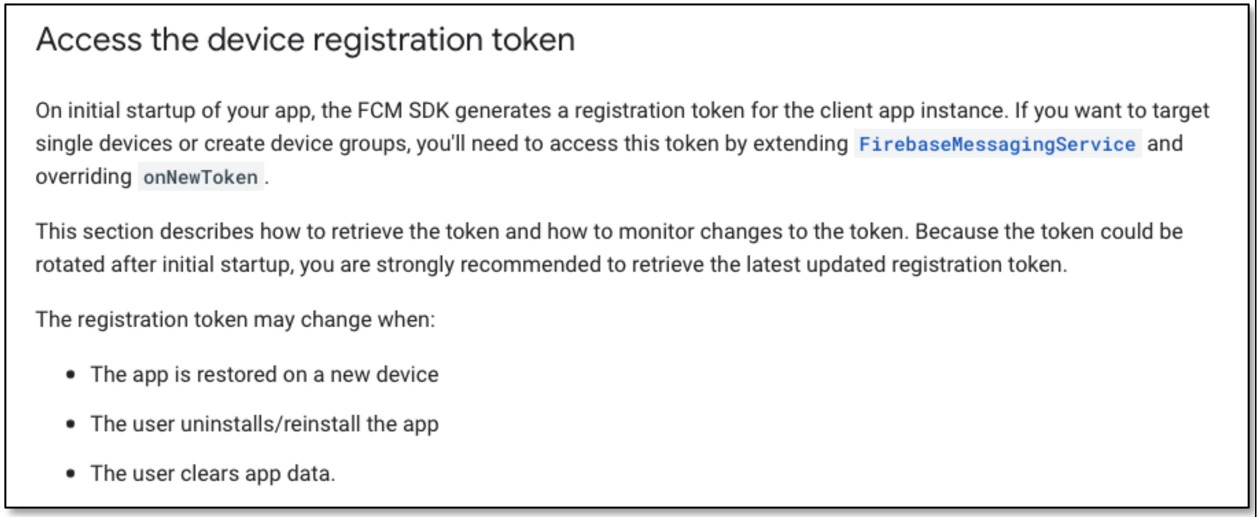
    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

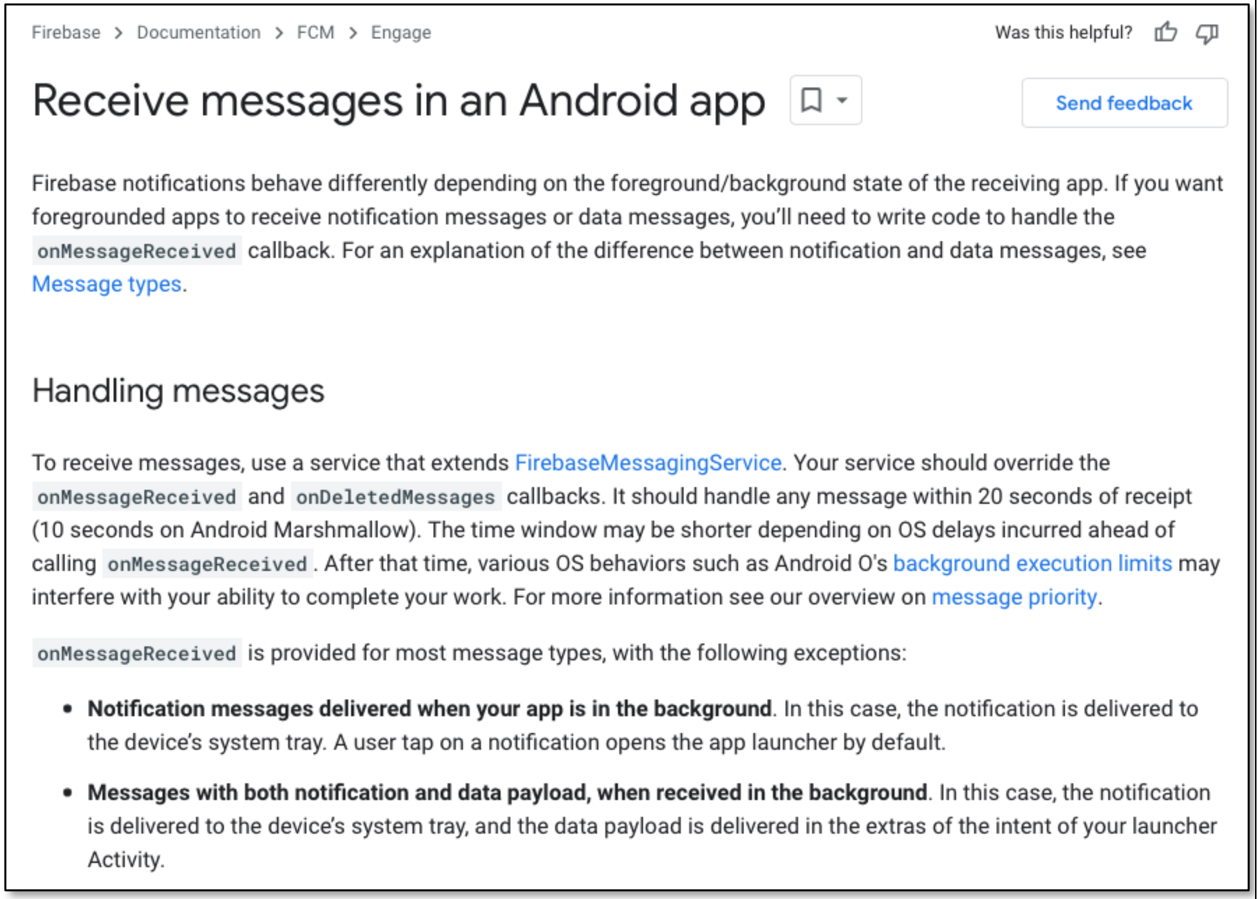
https://docs.tizen.org/application/web/guides/messaging/push/.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, decrypt

	messages from a plurality of servers which have identifiers and content for delivery to particular device agents.  Firewall Active Directory Windows devices Chrome OS devices ios devices SAMSUNG Knox Manage Cloud Connector Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 **Components of Knox Manage** 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
1[f] based on the particular agent identifier, deliver the message content to the particular device agent over the agent communication bus.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices "based on the particular agent identifier, deliver the message content to the particular device agent over the agent communication bus." For example, Samsung Galaxy phones and tablets deliver message content to particular device agents based on identifier. *See e.g.*,

	 Access the device registration token On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`. This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token. The registration token may change when: - The app is restored on a new device - The user uninstalls/reinstall the app - The user clears app data. https://firebase.google.com/docs/cloud-messaging/android/client;



Receive messages in an Android app

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see Message types.

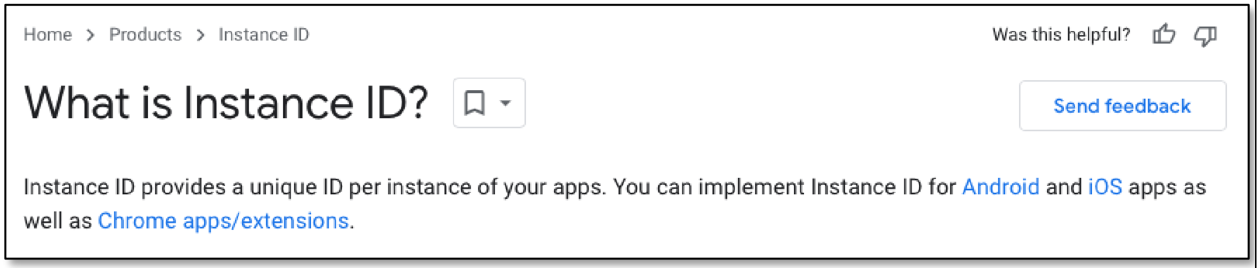
Handling messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's background execution limits may interfere with your ability to complete your work. For more information see our overview on message priority.

`onMessageReceived` is provided for most message types, with the following exceptions:

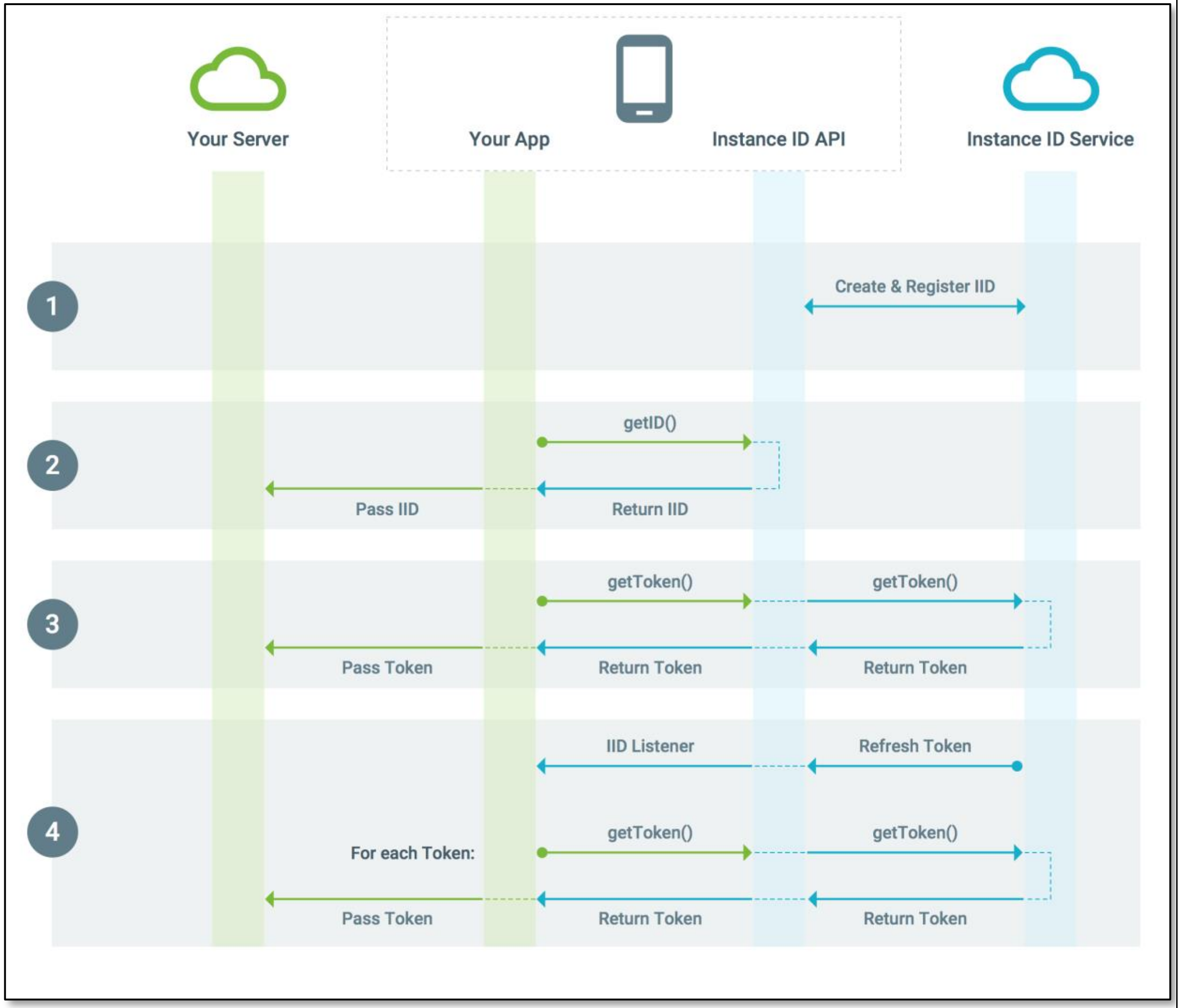
- **Notification messages delivered when your app is in the background**. In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

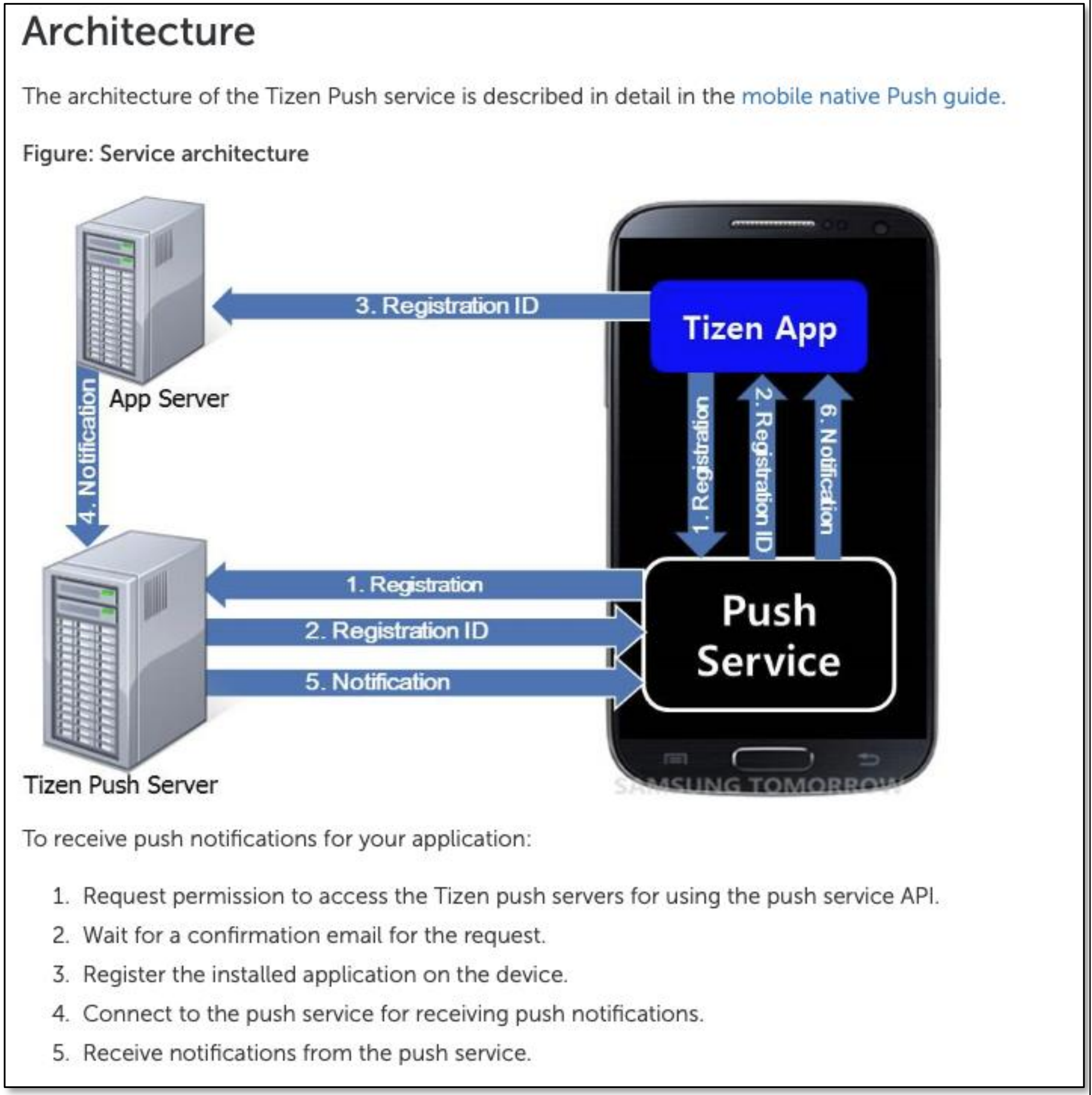
https://firebase.google.com/docs/cloud-messaging/android/receive;



What is Instance ID?

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for Android and iOS apps as well as Chrome apps/extensions.

	 Your Server Your App Instance ID API Instance ID Service 1 Create & Register IID 2 getID() Pass IID Return IID 3 getToken() getToken() Pass Token Return Token Return Token 4 IID Listener Refresh Token For each Token: getToken() getToken() Pass Token Return Token Return Token https://developers.google.com/instance-id. For further example, Samsung's Tizen based devices deliver message content to particular device agents based on identifier. *See e.g.*,



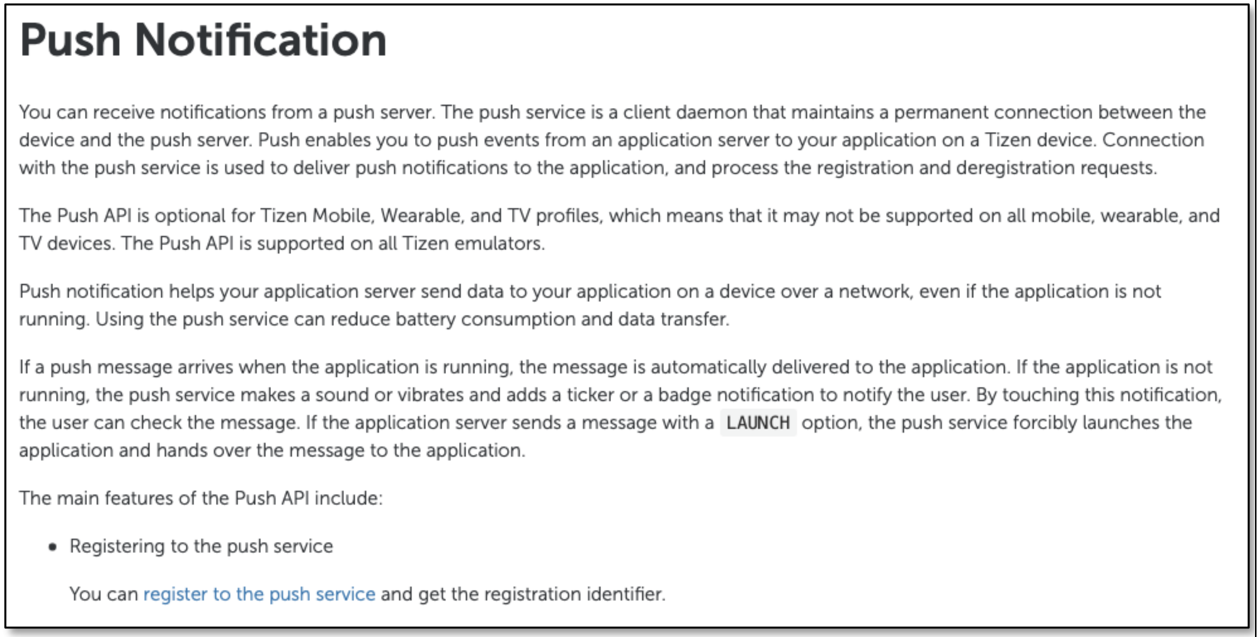
Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture

To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.



Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

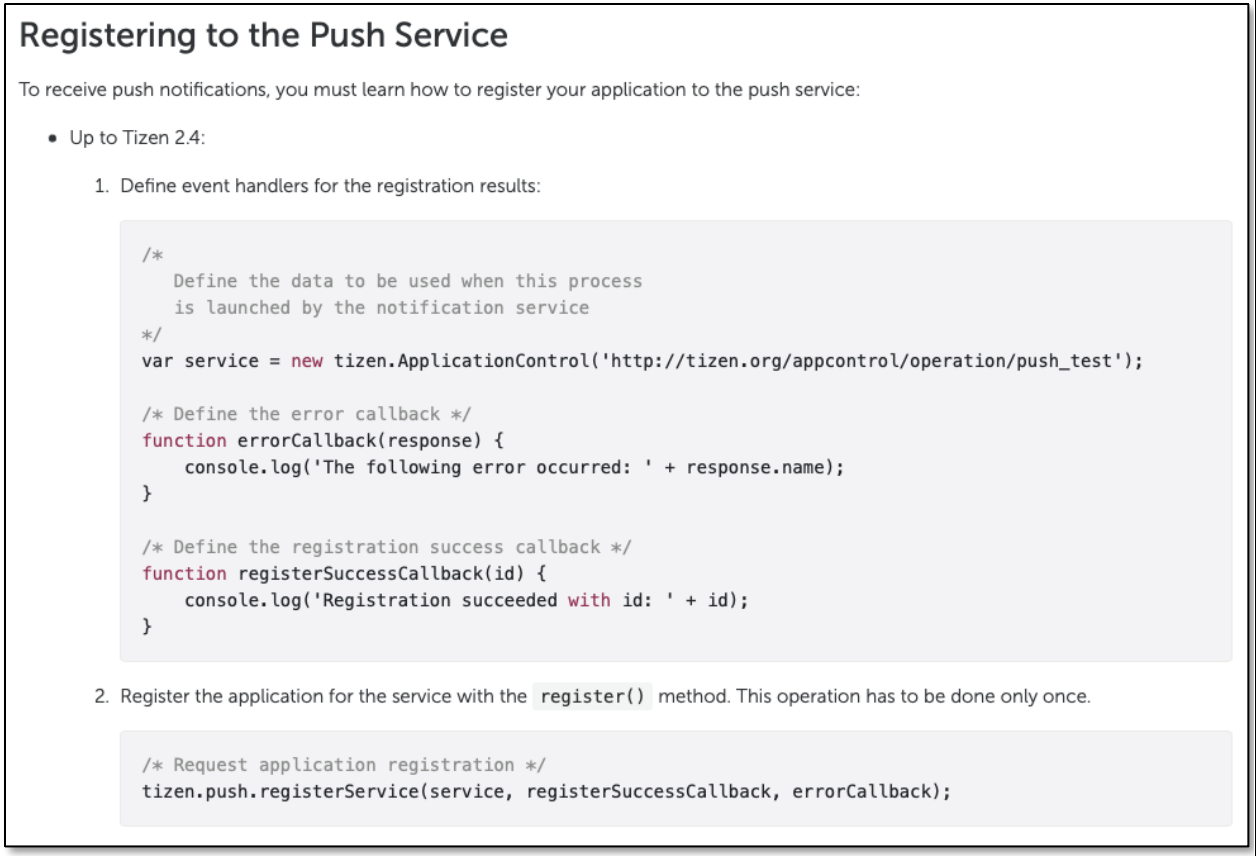
Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

  You can register to the push service and get the registration identifier.



Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

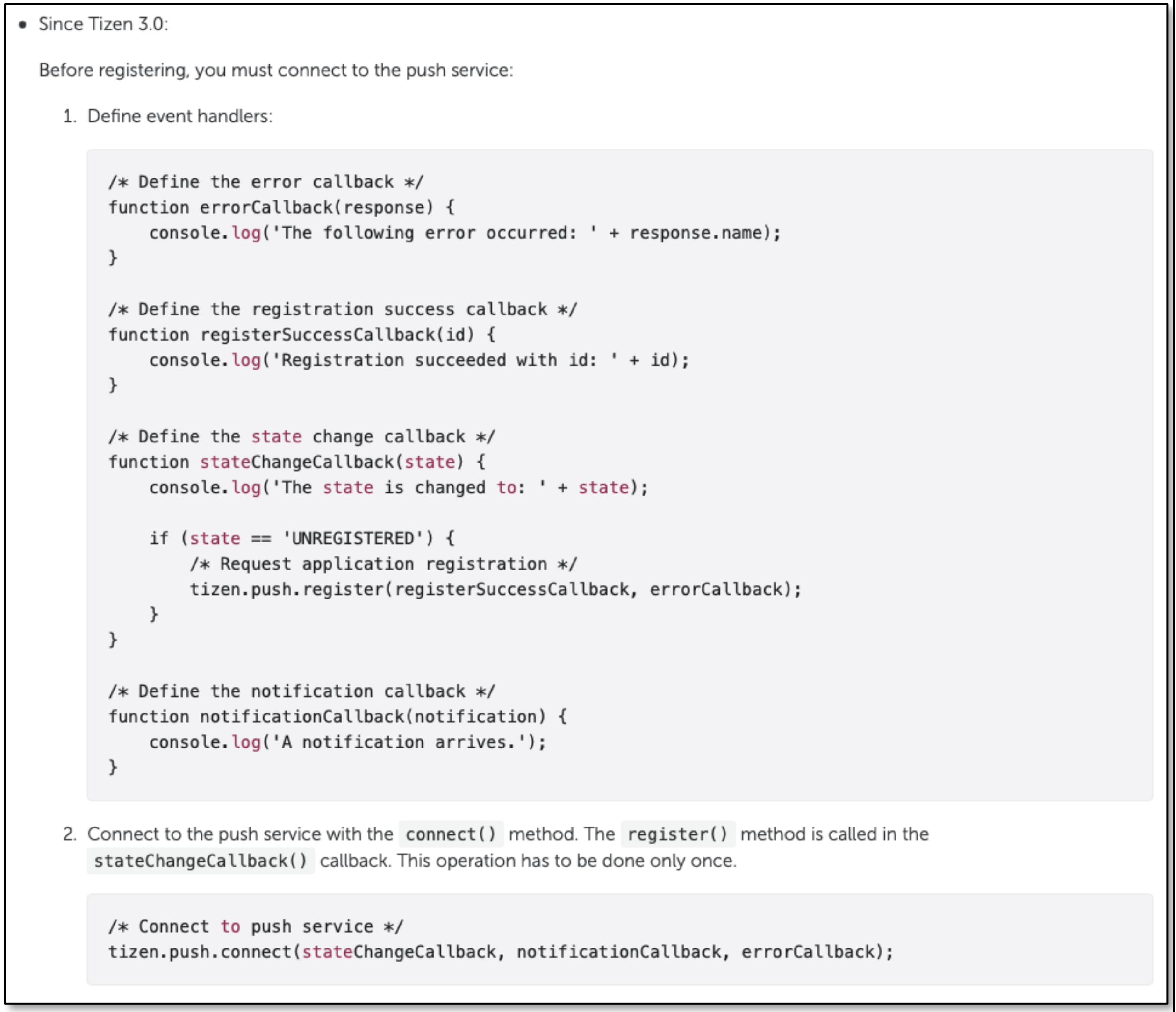
- Up to Tizen 2.4:

1. Define event handlers for the registration results:

```
/*
   Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```



- Since Tizen 3.0:

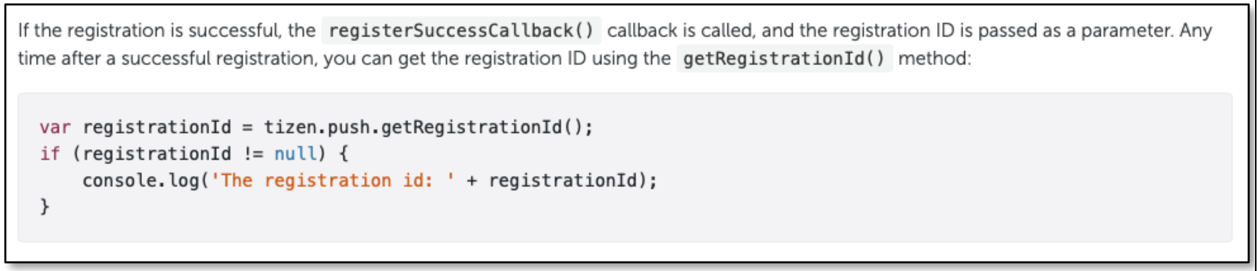
  Before registering, you must connect to the push service:

  1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state == 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
}
```

  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

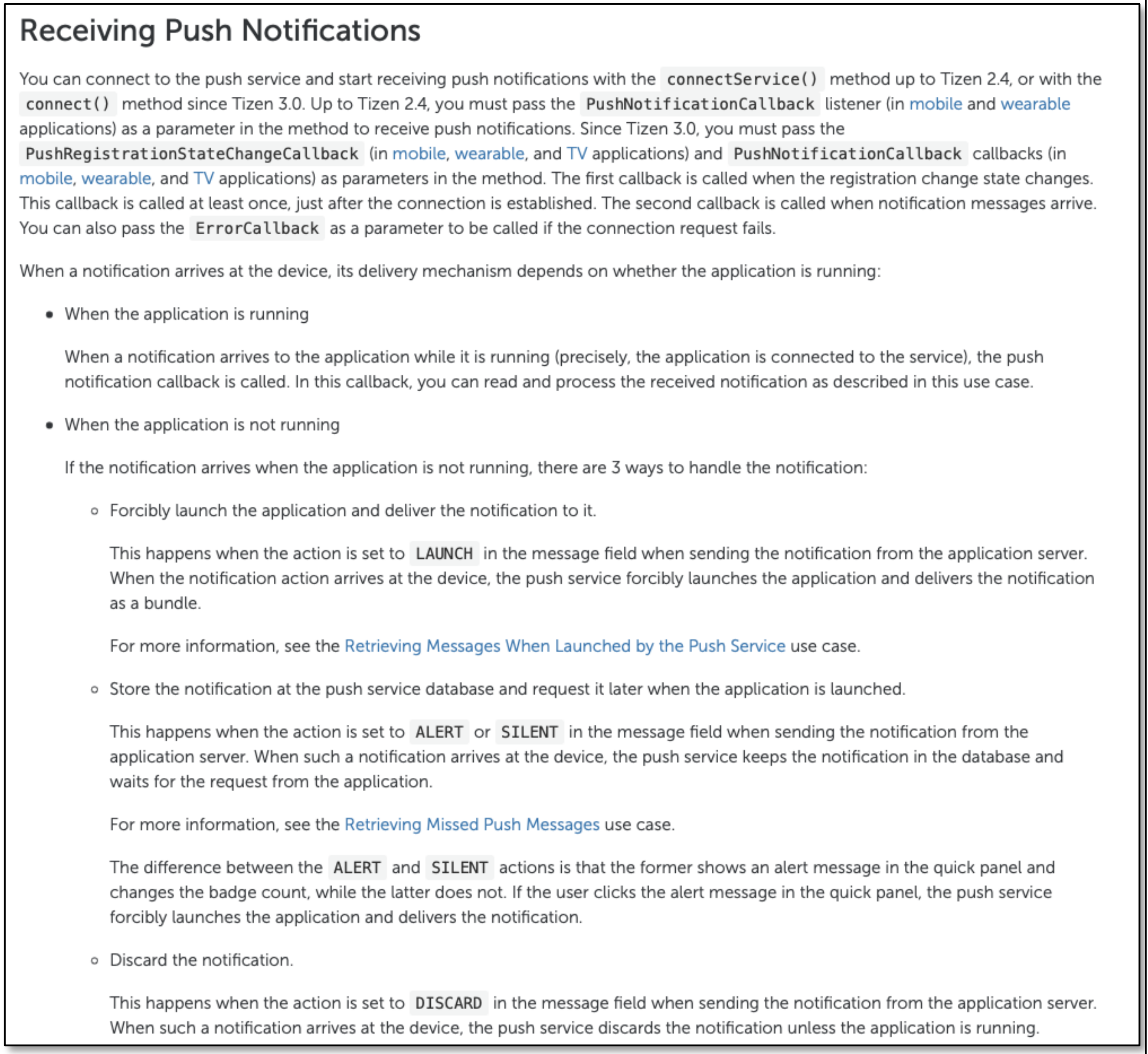
```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```



If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```

decrypt messages from a plurality of servers which have identifiers and content for delivery to particular device agents.



Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in mobile and wearable applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the `PushRegistrationStateChangeCallback` (in mobile, wearable, and TV applications) and `PushNotificationCallback` callbacks (in mobile, wearable, and TV applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

  When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

  If the notification arrives when the application is not running, there are 3 ways to handle the notification:

  - Forcibly launch the application and deliver the notification to it.

    This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

  - Store the notification at the push service database and request it later when the application is launched.

    This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

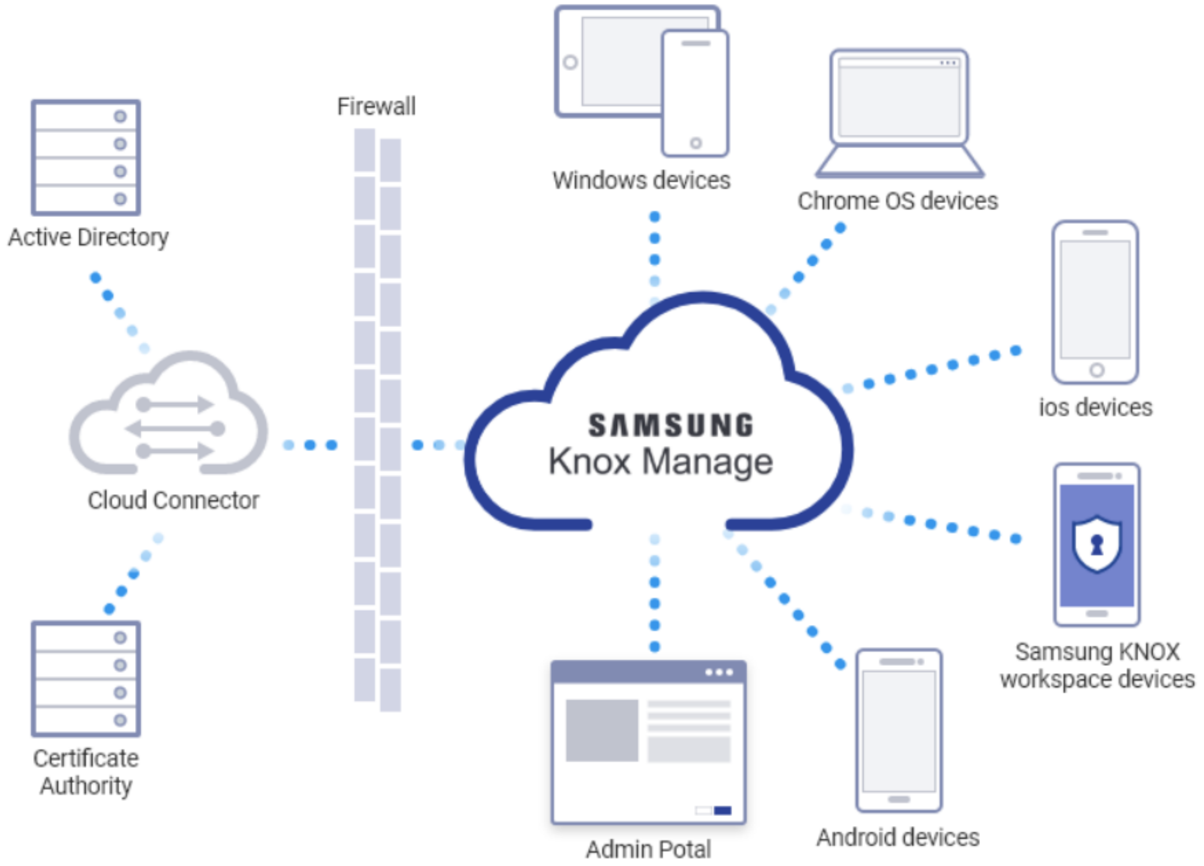
    For more information, see the Retrieving Missed Push Messages use case.

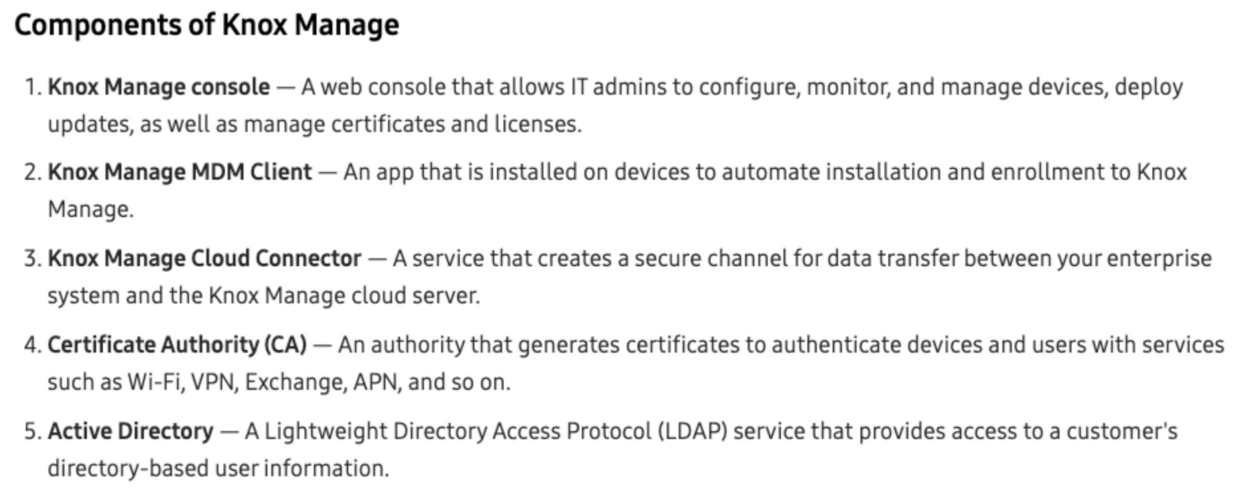
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

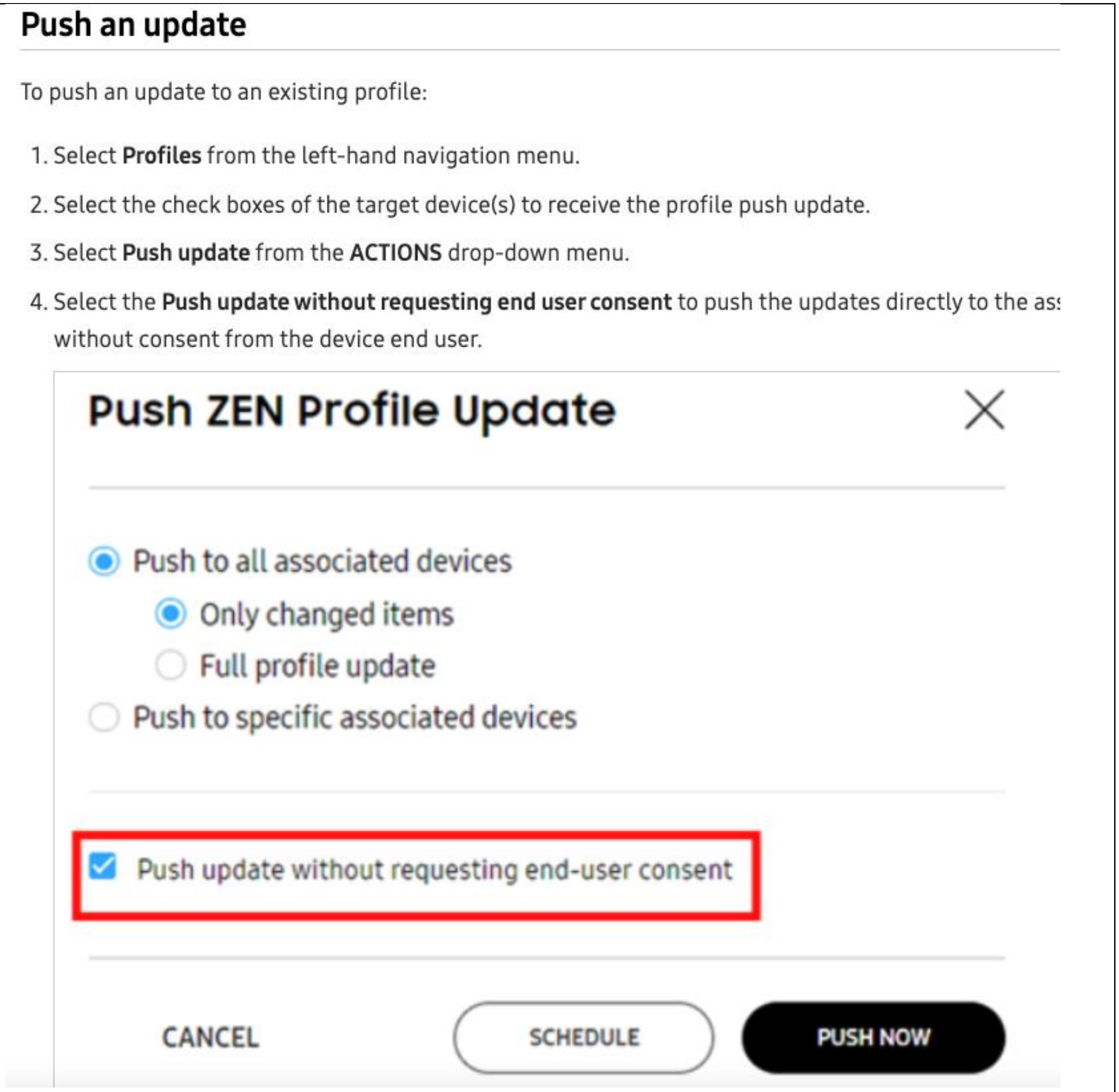
  - Discard the notification.

    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

https://docs.tizen.org/application/web/guides/messaging/push/.

	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, deliver message content to particular device agents based on identifier.  Firewall Windows devices Chrome OS devices Active Directory ios devices SAMSUNG Knox Manage Cloud Connector Samsung KNOX workspace devices Certificate Authority Admin Potal Android devices https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 **Components of Knox Manage** 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm



Push an update

To push an update to an existing profile:

1. Select **Profiles** from the left-hand navigation menu.
2. Select the check boxes of the target device(s) to receive the profile push update.
3. Select **Push update** from the **ACTIONS** drop-down menu.
4. Select the **Push update without requesting end user consent** to push the updates directly to the ass without consent from the device end user.

Push ZEN Profile Update

● Push to all associated devices

● Only changed items

○ Full profile update

○ Push to specific associated devices

☑ Push update without requesting end-user consent

https://docs.samsungknox.com/admin/knox-configure-wearables/update-existing-profile.htm